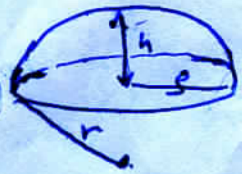
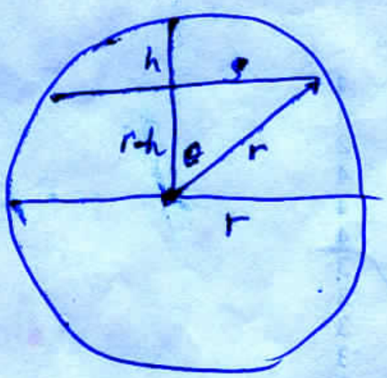
Reptory's pg 147 — March 89 – Tiling A Sphere — MRNQ39



Area of spherical cap $= 2\pi r h$

θ = colatitude



$\cos\theta = \frac{r-h}{r}$ $\qquad h=0 \qquad\qquad h=r$

$\theta = \cos^{-1}\frac{r-h}{r}$ $\qquad \theta = \cos^{-1}(1) = 0 \qquad \theta = \cos^{-1}(0) = \frac{\pi}{2}$

$\rho^2 + (r-h)^2 = r^2$

$\rho^2 = r^2 - (r-h)^2$ $\qquad h=0 \qquad\qquad h=r$

$= r^2 - r^2 - h^2 + 2rh$ $\qquad \rho^2 = 0 \qquad \rho^2 = r(2r-r) = r^2$

$= 2rh - h^2$

$= h(2r-h)$

---

circumference of bottom of cap $= 2\pi\rho$

total area of sphere $= A_s = 4\pi$

Area of

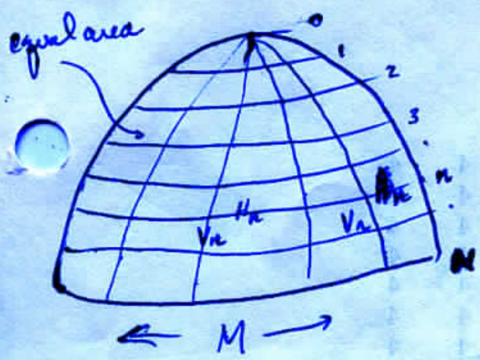
0	1	2	3	4
m	2	50	50	

$\frac{65535}{65535}$

128 64 8

62

31



Area of whole N. Hemisphere: 2π

Area of cap from N pole to division n: $\frac{2\pi n}{N}$

h of cap: since $A = 2\pi h$

$$h = \frac{A}{2\pi} = \frac{n}{N}$$

ρ of cap: $\rho = \sqrt{h(2-h)}$

circumference of bottom of cap $= 2\pi\rho$

length of one horizontal segment $\frac{2\pi\rho}{M}$

$$\text{Horizontal}_n = \frac{2\pi}{M}\sqrt{\frac{n}{N}\left(2-\frac{n}{N}\right)} \qquad n = 0, 1, 2 \ldots N$$

Area of n^{th} cap $\frac{2\pi n}{N}$, $h_n = \frac{n}{N}$, $\rho_n = \sqrt{\frac{n}{N}\left(2-\frac{n}{N}\right)}$

Area of $(n-1)^{th}$ cap $\frac{2\pi(n-1)}{N}$, $h_{n-1} = \frac{(n-1)}{N}$, $\rho_{n-1} = \sqrt{\frac{(n-1)}{N}\left[2-\frac{(n-1)}{N}\right]}$

angles $\theta_n = \cos^{-1}\left[1-\frac{n}{N}\right]$ $\theta_{n-1} = \cos^{-1}\left[1-\frac{(n-1)}{N}\right]$

difference in angles $\theta_n - \theta_{n-1}$ which is length of vertical segment $n-1$

$$\text{Vertical}_n = \cos^{-1}\left(1-\frac{n+1}{N}\right) - \cos^{-1}\left(1-\frac{n}{N}\right) \qquad n = 0 \ldots N-1$$

#ifdef

define N (7)

define M (16)

define N_ROW (M)

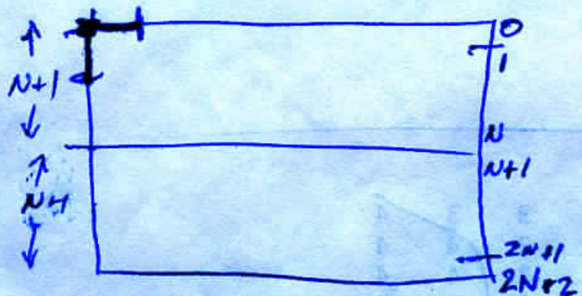
define N_COL (2*N+2)

double DA ((4.0*3.1415926*6371.0*6371.0/(2.0*N*M)));

double Vlen[N+1], Hlen[N+1];

for (i=0; i<=N; i++)
Hlen[i] = horizontal_n

for (i=0; i<=(N-1); i++)
Vlen[i] = vertical_n



Vertical: Cell

if y<1
RETURN φ

else if y>= && y<

y>=1 && y<=(N+1)

n = y-1

Return vert_n or hor_n

y>=(N+1) && y<=(2N+1)

n = (2N+1) - y

y=1 n=0
y=N+1

case horizontal

y>=1 && y<=(N+1)
n = y-1
rtn H_n

(y=1 n=0
y=N+1 n=N)

y>=(N+1) && y<=(2N+1)

n = (2N+1) - y
rtn H_n

(y = 2N+1 n=0
y = N+1 n = 2N+1-N-1
= N)

else
rtn 0.0

case Vertical

y>=1 && y<=N
n = y-1
RTN V_n

(y=1 n=0
y=N n=N-1)

y>=(N+1) && y<=2N
n = 2N-y

(y=2N n=0
y=N+1 n=2N-N-1 = N-1)

```
  4   4   1   1   1   1   1   1   1   1   1   4   4   4   4   4
  4   4   1   1   1   1   1   1   1   1   1   1   4   4   4   4
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

```
 plate area circumference
0 44.000000 36.000000
1 70.000000 44.000000
2 49.000000 38.000000
3 61.000000 44.000000
```

```
  4   2   2   2   2   2   2   2   2   2   2   2   2   2   4   4
  4   2   2   2   2   2   2   2   2   2   2   2   2   2   4   4
  2   2   2   2   2   4   4   2   2   2   2   4   4   4   4   2
  2   2   2   2   2   4   4   4   2   2   4   4   4   4   4   4
  2   2   2   1   1   1   4   4   4   4   4   4   4   4   4   4
  1   1   1   1   1   1   4   4   4   4   4   4   4   4   4   4
  1   1   1   1   1   1   1   4   1   4   4   4   4   4   4   3
  1   1   1   1   1   1   1   1   1   1   3   3   3   3   3   3
  3   1   1   1   1   1   1   1   1   1   3   3   3   3   3   3
  3   3   1   1   1   1   1   1   1   1   3   3   3   3   3   3
  3   1   1   1   1   1   1   1   1   3   3   3   3   3   3   3
  3   3   3   1   1   1   1   1   1   3   3   3   3   3   3   3
  3   3   1   1   1   1   1   1   1   3   3   3   3   3   3   3
  3   3   1   1   1   1   1   1   1   1   1   1   3   3   3   3
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

```
 plate area circumference
0 75.000000 52.000000
1 46.000000 44.000000
2 55.000000 38.000000
3 48.000000 46.000000
```

```
  2   2   1   1   1   1   1   1   1   1   1   1   1   1   1   2
  2   2   2   1   1   1   1   1   1   1   1   1   1   1   1   2
  2   2   2   2   2   1   2   2   1   1   1   1   1   1   2   2
  2   2   2   2   2   2   2   2   2   2   1   1   1   1   2   2
  2   2   4   4   2   2   2   2   2   2   1   1   1   3   2   2
  4   4   4   4   2   2   2   2   2   2   1   1   3   3   3   4
  4   4   4   4   4   2   2   2   2   3   1   1   3   3   4   4
  4   4   4   4   4   2   2   2   3   3   1   3   3   3   3   4
  4   4   4   4   4   4   2   2   3   3   3   3   3   3   3   4
  4   4   4   4   4   2   2   2   3   3   3   3   3   3   3   4
  4   4   4   4   2   2   2   3   3   3   3   3   3   3   3   4
  4   4   4   4   2   2   3   3   3   3   3   3   3   3   3   4
  4   4   4   4   4   4   3   3   3   3   3   3   3   3   4   4
  4   4   4   4   4   4   3   3   3   3   3   3   3   3   3   4
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

```
 plate area circumference
0 44.000000 44.000000
1 61.000000 56.000000
2 61.000000 46.000000
3 58.000000 42.000000
```

```
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   1   3   3   3   3   3   1   4   4   3   3   3
  3   1   1   1   1   1   1   1   1   1   1   4   4   3   4   3
  1   1   1   1   1   1   1   1   1   1   4   4   4   4   4   1
```

```
0 66.000000 44.000000
1 38.000000 34.000000
2 60.000000 66.000000
3 60.000000 56.000000
```

```
  4   4   4   4   4   4   4   4   4   2   2   2   2   4   4   4
  4   4   4   4   4   4   4   2   2   2   2   2   4   4   4   4
  4   4   4   4   4   4   4   2   2   2   2   2   4   4   4   4
  4   4   4   4   4   4   2   2   2   2   2   2   2   2   4   4
  4   4   4   4   4   1   2   2   2   2   2   2   2   2   4   4
  4   4   1   1   1   1   2   2   2   2   2   2   2   4   4   4
  4   1   1   1   1   1   2   2   2   2   2   2   2   3   4   4
  4   1   1   1   1   1   2   2   2   2   2   2   2   3   3   4
  1   1   1   1   1   1   2   2   2   2   2   2   3   3   1   1
  1   1   1   1   1   1   1   2   2   2   2   3   3   3   3   1
  1   1   1   1   1   1   1   1   3   3   3   3   3   3   3   1
  3   3   3   1   1   1   1   1   3   3   3   3   3   3   3   3
  3   3   3   1   1   1   1   1   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

```
 plate area circumference
0 50.000000 38.000000
1 61.000000 38.000000
2 54.000000 48.000000
3 59.000000 44.000000
```

```
  1   1   1   1   1   1   1   1   2   2   2   2   2   2   1   1
  4   1   4   4   1   1   1   1   1   2   2   2   2   2   1   1
  4   4   4   4   4   4   1   1   2   2   2   2   2   4   1   1
  4   4   4   4   4   1   1   1   1   2   2   2   2   4   4   4
  4   4   4   4   4   4   1   1   1   1   2   2   4   4   4   4
  4   4   4   4   4   4   1   1   1   1   2   2   4   4   4   4
  3   3   4   4   4   4   1   1   1   2   2   2   4   4   4   4
  3   3   3   4   3   3   1   1   1   2   2   2   4   4   3   3
  3   3   3   4   3   3   1   2   2   2   2   2   4   4   3   3
  3   3   3   3   3   3   2   2   2   2   2   2   4   3   3   3
  3   3   3   3   3   3   3   2   2   2   2   3   3   3   3   3
  3   3   3   3   3   3   3   3   2   2   2   3   3   3   3   3
  3   3   3   3   3   3   3   3   2   2   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   2   2   3   3   3   3   3   3
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

```
 plate area circumference
0 41.000000 52.000000
1 52.000000 50.000000
2 78.000000 48.000000
3 53.000000 50.000000
```

```
  3   2   2   2   2   2   2   2   2   2   2   2   3   3   3   3
  3   2   2   2   2   2   2   2   2   2   2   3   3   3   3   3
  3   3   2   2   2   2   2   2   2   2   2   3   3   3   3   3
  3   2   2   2   2   2   2   2   2   2   2   3   3   3   3   3
  3   2   2   2   2   2   2   2   2   2   2   4   3   3   3   3
  3   3   2   2   2   2   2   2   2   2   4   4   3   3   3   3
  3   3   2   2   2   2   2   2   4   4   4   4   4   3   3   3
  3   3   3   3   2   2   2   1   1   4   4   4   4   3   3   3
  4   3   3   2   2   2   1   1   1   4   4   4   4   4   4   4
  4   4   4   1   1   1   1   1   1   4   4   4   4   4   4   4
  4   4   4   1   1   1   1   1   1   1   4   4   4   4   4   4
  4   4   4   4   1   1   1   1   1   1   1   4   4   4   4   4
```

3	3	3	3	3	3	4	4	4	4	4	4	4	4	3	3
3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	3
3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	3
3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	3
3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	3
3	3	3	3	3	3	4	4	4	4	4	1	1	1	3	3
3	3	3	2	2	4	4	4	4	4	4	1	1	1	3	3
3	3	2	2	2	2	4	4	4	4	1	1	1	1	1	3
2	2	2	2	2	2	2	2	4	1	1	1	1	1	1	3
1	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
1	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1
1	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1
1	1	2	2	2	2	2	2	1	1	1	1	1	1	1	1
1	1	2	2	2	2	2	1	1	1	1	1	1	1	1	1
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

plate	area	circumference
0	64.000000	40.000000
1	47.000000	34.000000
2	53.000000	36.000000
3	60.000000	38.000000

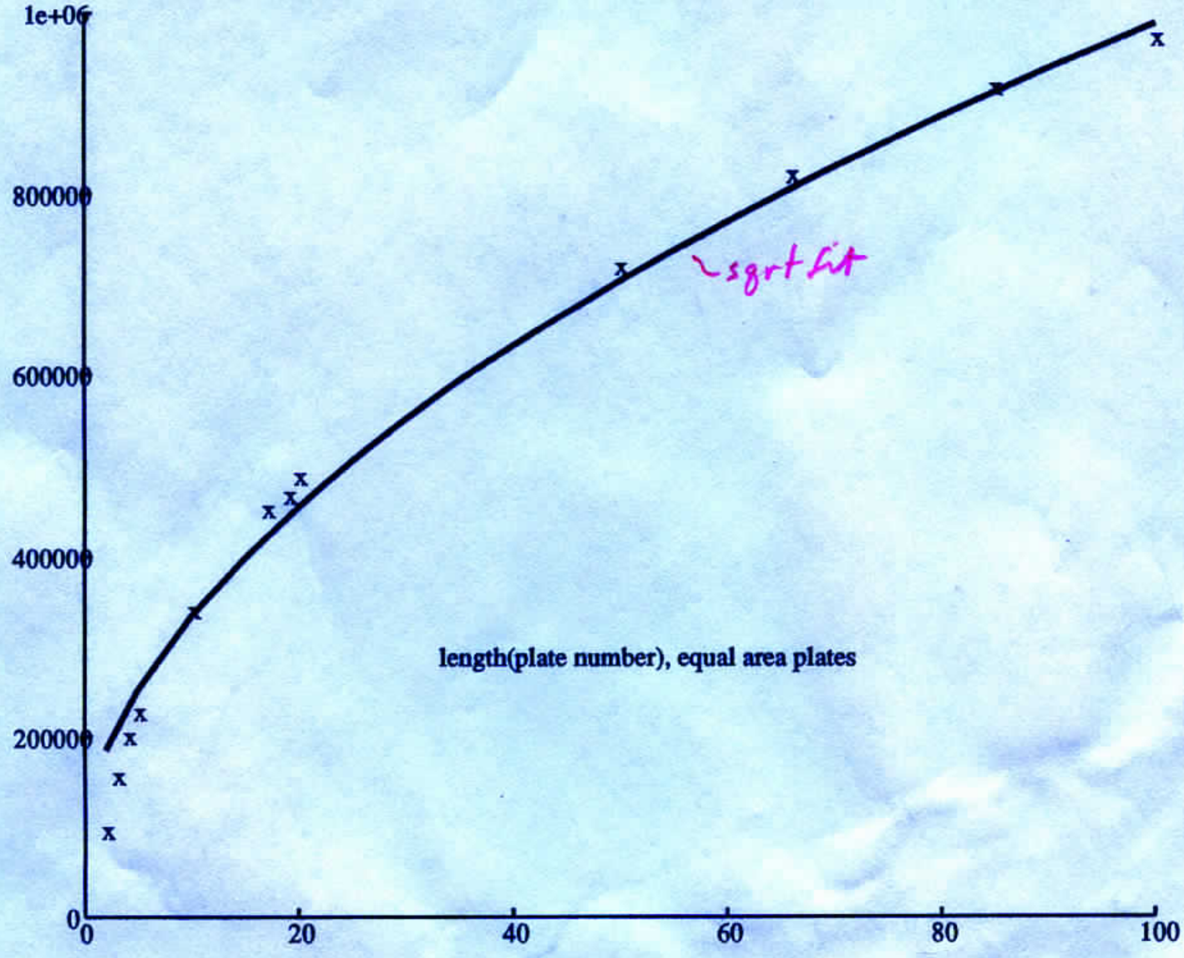
4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	4
4	4	4	4	4	4	3	3	3	3	3	3	3	3	4	4
4	4	4	4	4	4	3	3	3	3	3	3	3	3	4	4
4	4	4	4	4	4	3	3	3	3	3	3	3	3	4	4
4	4	4	4	4	4	3	3	3	3	3	3	3	3	4	4
4	4	4	4	4	4	4	3	3	3	3	3	3	3	2	4
4	4	4	4	4	4	4	3	3	3	3	2	2	2	2	2
4	4	4	4	4	4	4	3	3	3	2	2	2	2	2	2
2	4	4	4	1	1	1	3	2	2	2	2	2	2	2	2
1	1	1	4	1	1	1	1	2	2	2	2	2	2	2	1
1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	1
1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	1
1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	1
1	1	1	1	1	1	1	1	2	2	2	2	2	1	1	1
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

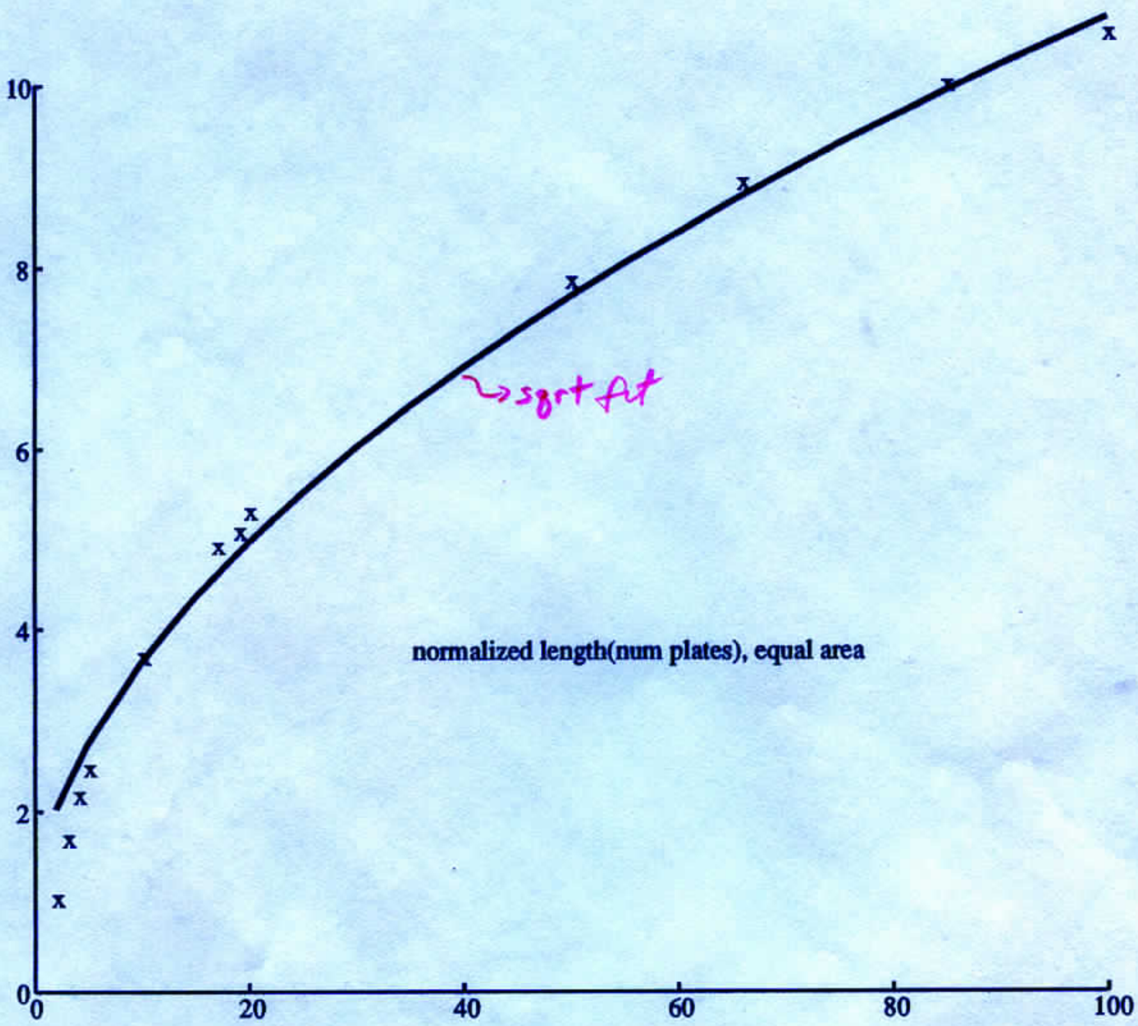
plate	area	circumference
0	49.000000	36.000000
1	54.000000	36.000000
2	56.000000	36.000000
3	65.000000	38.000000

1	1	1	1	1	1	1	3	3	3	3	4	4	4	4	4
1	1	1	1	1	1	1	1	1	1	3	3	3	4	4	4
1	1	1	1	1	1	1	1	3	3	3	3	3	3	4	1
1	1	1	1	1	1	1	1	1	3	3	3	3	3	4	4
1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	4
4	1	1	1	1	1	1	1	1	1	3	3	3	3	4	4
4	4	1	1	1	1	1	1	3	3	3	3	3	3	4	4
4	4	4	1	1	1	1	2	2	3	3	3	3	3	4	4
4	4	4	4	1	1	2	2	2	2	2	3	3	3	3	4
4	4	4	4	4	2	2	2	2	2	2	3	3	3	3	4
4	4	4	4	4	4	2	2	2	2	2	2	2	3	4	4
4	4	4	4	4	4	2	2	2	2	2	3	3	3	4	4
4	4	4	4	2	2	2	2	2	2	2	3	3	3	4	4
3	4	4	4	2	2	2	2	2	2	3	3	3	3	3	3
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

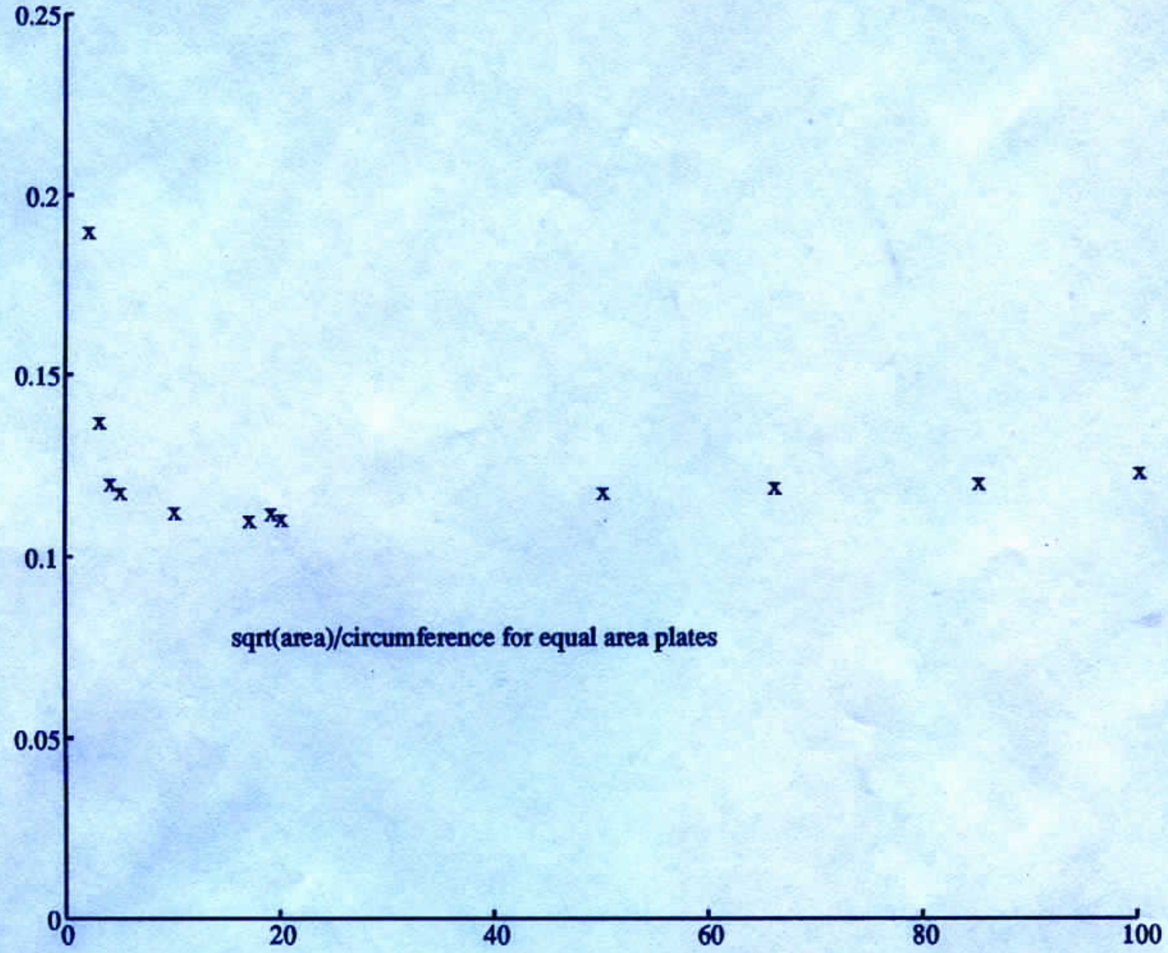
plate area circumference

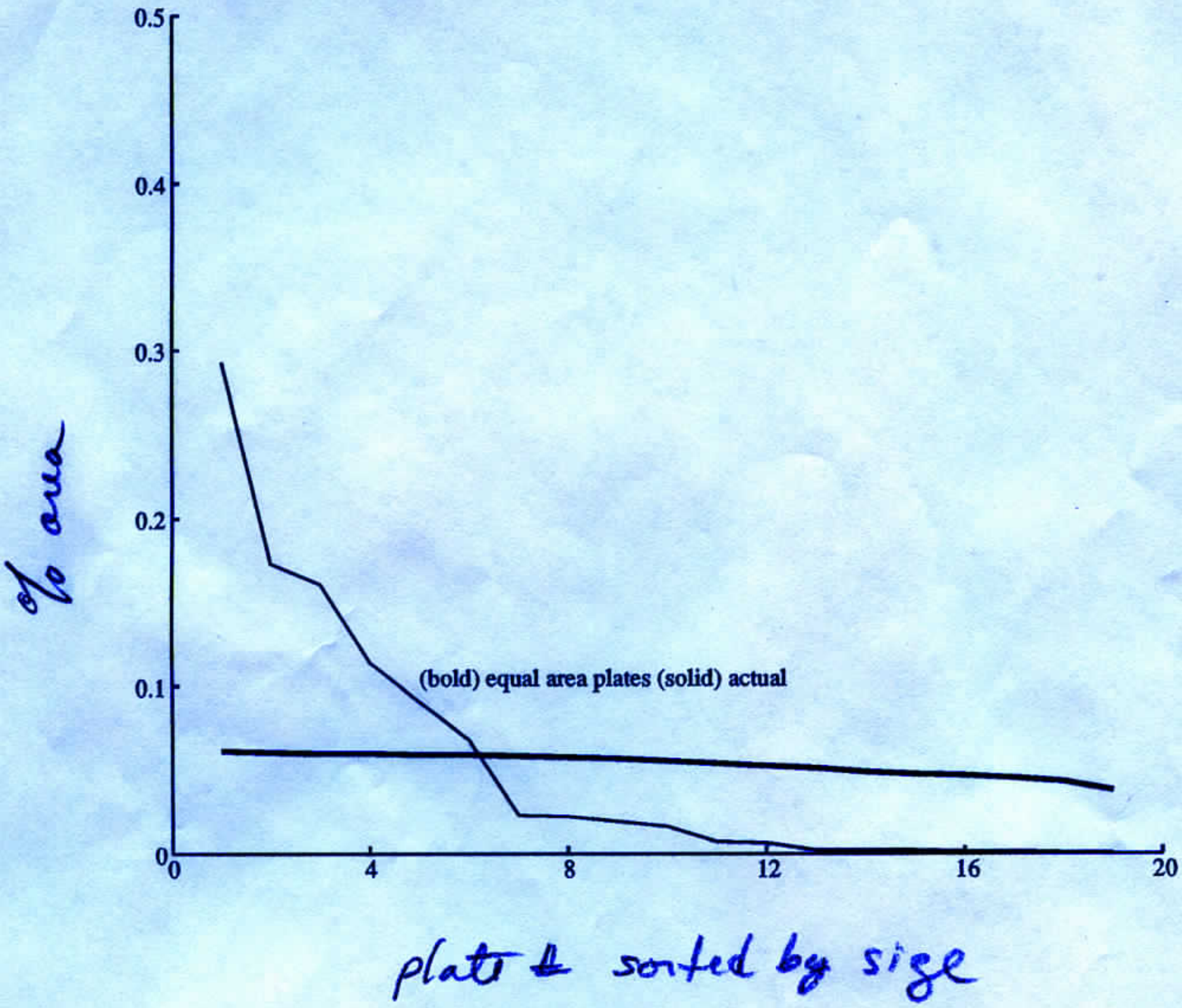
```
make_plates 5 20 20 20 20 20

(5, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 2, 2, 2, 2)
(2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 2, 2, 2)
(2, 3, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 1, 2, 2, 1)
(3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 5, 1, 1, 1, 1, 1)
(3, 3, 3, 3, 2, 4, 2, 2, 2, 2, 5, 1, 1, 1, 1, 3)
(3, 3, 3, 3, 2, 4, 4, 4, 2, 5, 5, 1, 1, 1, 1, 1)
(3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 1, 1, 1, 1, 1)
(1, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 1, 1, 1, 1, 1)
(1, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 1, 1, 1, 1, 1)
(1, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 1, 1, 1, 1, 1)
(3, 3, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5, 1, 1, 1, 1)
(3, 3, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5, 5, 1, 1, 4)
(4, 3, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5, 5, 5, 5, 4)
(4, 3, 3, 3, 3, 5, 4, 4, 4, 5, 5, 5, 5, 5, 4, 4)

(3, 3, 3, 3, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3)
(3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 4)
(4, 3, 3, 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4)
(4, 5, 3, 5, 4, 4, 2, 3, 3, 3, 2, 4, 2, 4, 4, 4)
(5, 5, 5, 5, 4, 4, 2, 2, 2, 2, 2, 2, 2, 4, 4, 4)
(4, 5, 5, 5, 5, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4)
(5, 5, 5, 5, 5, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4)
(5, 5, 5, 5, 5, 5, 5, 2, 2, 2, 2, 2, 2, 1, 1, 5)
(5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1)
(1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1)
(1, 5, 5, 5, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1)
(1, 1, 5, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1, 1)
(1, 1, 5, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1, 1)
(1, 1, 5, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 1, 1)

(5, 4, 4, 4, 4, 5, 5, 5, 5, 4, 4, 4, 4, 5, 5, 5)
(5, 4, 4, 4, 4, 5, 5, 5, 5, 4, 4, 4, 5, 5, 5, 5)
(5, 4, 4, 4, 4, 5, 5, 5, 5, 4, 4, 4, 5, 5, 5, 5)
(5, 5, 4, 4, 4, 4, 5, 5, 5, 4, 4, 4, 5, 5, 2, 2)
(5, 5, 5, 4, 4, 4, 5, 5, 5, 4, 4, 4, 5, 2, 2, 2)
(2, 1, 5, 4, 4, 4, 4, 5, 5, 4, 3, 3, 3, 3, 2, 2)
(1, 1, 5, 1, 1, 1, 5, 5, 5, 3, 3, 3, 3, 3, 2, 2)
(1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 2, 2, 2)
(1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 2, 1, 1)
(2, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2)
(2, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2)
(2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2)
(2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2)
(2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2)
```

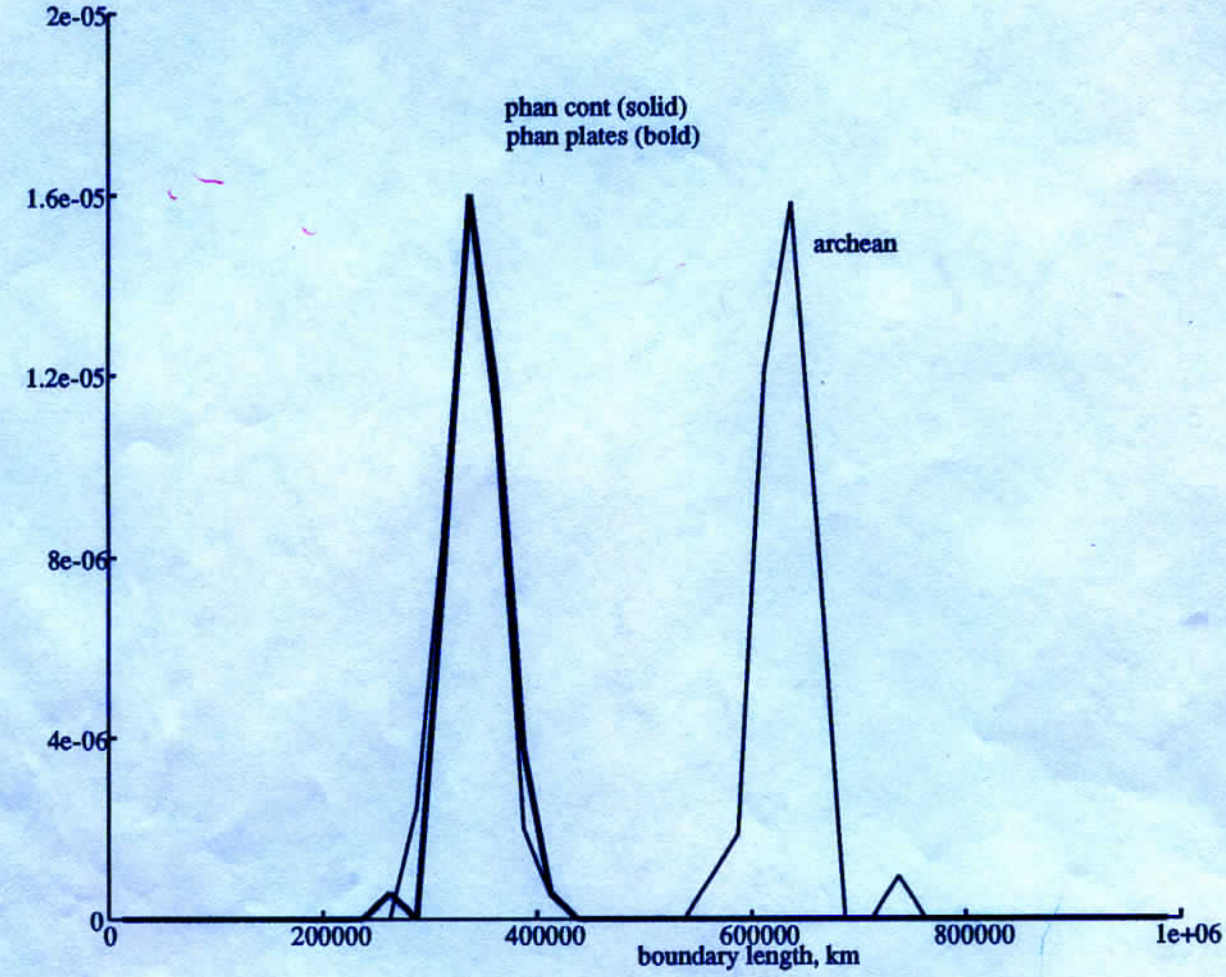

1e+06
800000
600000
400000
200000
0
0
20
40
60
80
100
length(plate number), equal area plates
~sqrt fit



10
8
6
4
2
0
0
20
40
60
80
100
normalized length(num plates), equal area
sqrt fit


0.25
0.2
0.15
0.1
0.05
0
0
20
40
60
80
100
sqrt(area)/circumference for equal area plates


0.5
0.4
0.3
0.2
0.1
0
% area
(bold) equal area plates (solid) actual
0
4
8
12
16
20
plate # sorted by size


2e-05
1.6e-05
1.2e-05
8e-06
4e-06
0
0
200000
400000
600000
800000
1e+06
boundary length, km
phan cont (solid)
phan plates (bold)
archean

```c
/* make_plates.c */

#include <stdio.h>
#include <curses.h>
#include <math.h>
char *malloc();

double compute_area(), compute_circumference();
double acos(), sqrt();
int compare_floats();

int N_PLATES;
#define N_PLATES_MAX (106)
#define N_REALIZATIONS (100)

/* this defines the size of the equal area grid */
/* 2N is the number of cells in vertical */
/* M is the number of cells in horizontal */
/* these numbers have been carefully tuned to give */
/* mean segment lengths of about 122 km in both hor and vert */
/* there are 46592 total bins */


#define N (81)
#define M (256)

/* test size */
/*
#define N (4)
#define M (16)
*/


#define N_COLS (M)
#define N_ROWS (2*N+2)
#define NULL_PLATE (0)
#define HORIZONTAL (TRUE)
#define VERTICAL (FALSE)
#define ACTIVE (TRUE)
#define INACTIVE (FALSE)

#define RADIUS (6371.0)
#define PI (3.1415926)

double dA = 4.0*PI*RADIUS*RADIUS/(2.0*N*M);

double vlen[N], hlen[N+1];
```

```
#define PHONEY_PLATE (255)

#define wrap(x)  (((x)<0)?(x+N_COLS):(((x)>=N_COLS)?(x-N_COLS):x))
#define slide(x)  wrap(((x)+(M/2)))
#define setmap(x,y,v)  (map[(y)][wrap(x)]=v)
#define getmap(x,y)  (map[(y)][wrap(x)])
#define ran_num(a,b)  ((((random()&65535)*(b+1-a))/65536)+a)

struct plate {
        struct segment *active; /*head of linked list of segments that may be replaced */
        struct segment *inactive; /*head of linked list of segments that are final */
        int active_count;
        int inactive_count;
        float area;
        float circumference
        };

struct plate the_plate[N_PLATES_MAX];

struct segment {
        struct segment *last;   /* order in linked list */
        struct segment *next;
        int orientation;        /* VERTICAL or HORIZONTAL */
        int x, y;               /* position */
        struct segment *left;   /* order in boundary, when standing in interior */
        struct segment *right;
        };

unsigned char map[N_ROWS][N_COLS];

int lookup[10000];
float percentage[N_PLATES_MAX];

int active_plates;

double total_area, total_length, original_total_percentage;

struct segment *insert_segment();

float ratio;
float fractional_area[N_PLATES_MAX];


main(argc, argv)
int argc;
char *argv[];
```

```
{
        int i, j, k, c;
        int x, y;
        int plate, seg_num;
        struct segment *seg;
        int found, realization;
        double mean_v, mean_h;

        /* read in args */

        if(argc < 2 ) {
                fprintf(stderr,"usage: make_plates n_plates percentage_1 percentage_2 ... percentage_n\n");
                exit(-1);
                }
        if( (sscanf(argv[1],"%d",&N_PLATES) != 1) || N_PLATES<1 || N_PLATES>=N_PLATES_MAX ) {
                fprintf(stderr,"error: bad n_plates: %s\ must be in range 1-%dn", argv[1], N_PLATES_MAX);
                exit(-1);
                }
        if( (argc-2) < N_PLATES ) {
                fprintf(stderr,"error: percentages for %d plates given, but %d expected\n", (argc-2), N_PLATES );
                exit(-1);
                }
        original_total_percentage=0.0;
        for( i=0; i<N_PLATES; i++ ) {
                if( (sscanf(argv[i+2],"%f",&(percentage[i])) != 1 ) || percentage[i]<=0.0 || percentage[i]>=100.0 ) {
                        fprintf(stderr,"error: percentage for plate %d bad: %s\n", i, argv[i+1] );
                        exit(-1);
                        }
                original_total_percentage+=percentage[i];
                }
        if( original_total_percentage<99.5 || original_total_percentage>100.5 ) {
                fprintf(stderr,"warning: percentages dont sum to 100, they sum to %f\n",
                        original_total_percentage);
                }

        /* initialize segment length arrays */
        mean_h=0.0;
        for( i=0; i<=N; i++ ) {
                double temp, NN, MM;
                NN = ((double)(N));
                MM = ((double)(M));
                temp = ((double)(i)) / NN;
                hlen[i] = RADIUS * 2.0 * PI * sqrt(temp*(2.0-temp)) / MM;
                mean_h += hlen[i];
                }
        mean_h = mean_h / ((double)(N+1));
```

```
	mean_v=0.0;
	for( i=0; i<N; i++ ) {
		double NN, MM, ii;
		NN = ((double)(N));
		MM = ((double)(M));
		ii = ((double)(i));
		vlen[i] = RADIUS * (acos( 1.0 - (ii+1.0)/NN ) - acos( 1.0 - ii/NN ));
		mean_v += vlen[i];
		}
	mean_v = mean_v / ((double)(N));


	/* initial random number generator */
	time( &i );
	srandom( i );

	printf("i\tlength\tratio\t");

	for( i=0; i<N_PLATES; i++ ) {
		if( i != (N_PLATES-1) ) {
			printf("area %d\t", i);
			}
		else {
			printf("area %d\n", i);
			}
		}

	for( realization=0; realization<N_REALIZATIONS; realization++ ) {


	/* initialize structures */
	active_plates = N_PLATES;
	for( i=0; i<N_ROWS; i++) for(j=0; j<N_COLS; j++) setmap(j,i,NULL_PLATE);
	for( j=0; j<N_COLS; j++ ) { setmap(j,0,PHONEY_PLATE); setmap(j,N_ROWS-1,PHONEY_PLATE); }


	for( i=0; i<N_PLATES; i++ ) {
		struct segment *s1, *s2, *s3, *s4;
		the_plate[i].active=NULL;
		the_plate[i].inactive=NULL;
		the_plate[i].active_count=0;
		the_plate[i].inactive_count=0;
		the_plate[i].area=0.0;
		the_plate[i].circumference=0.0;

		do {
			x = ran_num(0,N_COLS);
```

```
			y = ran_num(1,N_ROWS-1);
			} while ( getmap(x,y) != NULL_PLATE );
		setmap(x,y,i+1);
		s1=insert_segment( &(the_plate[i]), HORIZONTAL, x,   y   );
		s2=insert_segment( &(the_plate[i]), HORIZONTAL, x,   y+1 );
		s3=insert_segment( &(the_plate[i]), VERTICAL,   x,   y   );
		s4=insert_segment( &(the_plate[i]), VERTICAL,   x+1, y   );
		s1->left=s3; s1->right=s4;
		s3->left=s2; s3->right=s1;
		s2->left=s4; s2->right=s3;
		s4->left=s1; s4->right=s2;
		} /*end for*/

	set_lookup();

	c=0;
	while( active_plates != 0 ) {

		/* to avoid running out of memory, periodically cleanup */
		/* all interior segments */
		c++;
		if( c>1000 ) {
			cleanup_segments();
			c=0;
			}

		/* pick a radom plate and make sure that it has active segments */
		plate = lookup[ ran_num(0,9999) ];
		if( plate==PHONEY_PLATE || the_plate[plate].active==NULL ) {
			continue;
			}

		do {
			/* pick a random segment and find it in linked list */
			seg_num = ran_num(0,the_plate[plate].active_count-1);
			seg = the_plate[plate].active;
			i=0; found=FALSE;
			while( seg != NULL ) {
				if( i == seg_num ) {
					found=TRUE;
					break;
					}
				i++;
				seg=seg->next;
				} /* end while */
			if( !found ) {
				fprintf( stderr, "error: cant find seg %d plate %d\n", seg_num, plate);
```

```
                                exit(-1);
                                }
                        } while( !process_segment( seg, &(the_plate[plate]) ) );

                } /* end while */


        /* add up areas in each plate*/
        for( i=0; i<N_ROWS; i++) for( j=0; j<N_COLS; j++ ) {
                k=getmap(j,i)-1;
                if( k>=0 && k<N_PLATES ) {
                        the_plate[k].area += compute_area(j,i);
                        }
                }

        /* add up circumferences in each plate */
        for( i=0; i<N_PLATES; i++ ) {
                struct segment *current;
                current = the_plate[i].inactive;
                while( current!=NULL ) {
                        the_plate[i].circumference += compute_circumference(current->x, current->y, current->orientation);
                        current=current->next;
                        }
                }

        /* tally up total area and length */
        total_area=0.0;
        total_length = 0.0;
        ratio = 0.0;
        for( i=0; i<N_PLATES; i++ ) {
                total_area += the_plate[i].area;
                total_length += the_plate[i].circumference;
                ratio += sqrt(the_plate[i].area) / the_plate[i].circumference;
                }
        total_length = total_length / 2.0; /*since same line shared by two plate boundaries */
        ratio = ratio / N_PLATES;

        for( i=0; i<N_PLATES; i++ ) {
                fractional_area[i] = the_plate[i].area / total_area;
                }

        qsort( fractional_area, N_PLATES, sizeof(float), compare_floats );


        printf("%d\t%.2f\t%.3f\t", realization, total_length, ratio );
        for( i=0; i<N_PLATES; i++ ) {
                if( i != (N_PLATES-1) ) {
```

```
                        printf("%.3f\t", fractional_area[i] );
                        }
                else {
                        printf("%.3f\n", fractional_area[i] );
                        }
                }
        fflush(stdout);

        for( i=0; i<N_PLATES; i++ ) {
                free_up_list( &(the_plate[i].inactive) );
                }

        }
        }

int compare_floats(a,b)
float *a, *b;
{
        if( *a < *b ) {
                return(1);
                }
        else if( *a > *b ) {
                return(-1);
                }
        else {
                return(0);
                }
        }

double compute_area(x,y)
int x,y;
{
        return(dA);
        }

double compute_circumference(x,y,orientation)
int x,y;
{
        int n;
        if( orientation==HORIZONTAL ) {
                if( y>=1 && y<=(N+1) ) {
                        n=y-1;
                        return( hlen[n] );
                        }
                else if( y>=(N+1) && y<=(2*N+1) ) {
                        n=(2*N+1)-y;
                        return( hlen[n] );
```

```
				}
			else {
				return(0.0);
				}
			}
	else /* orientation == VERTICAL */ {
			if( y>=1 && y<=N ) {
				n = y-1;
				return( vlen[n] );
				}
			else if( y>=(N+1) && y<=(2*N) ) {
				n = 2*N - y;
				return( vlen[n] );
				}
			else {
				return( 0.0 );
				}
			}
	}

process_segment( seg, plate )
struct segment *seg;
struct plate *plate;
{
	int top, bottom, left, right, xnew;
	struct segment *s1, *s2, *s3;

	if( seg->orientation == HORIZONTAL ) {
			if( seg->y==1 ){ /*top of map*/
				xnew=slide(seg->x);
				top=getmap(xnew,1);
				bottom=getmap(seg->x,seg->y);
				}
			else if( seg->y == (N_ROWS-1) ) {
				xnew=slide(seg->x);
				top=getmap(seg->x,seg->y-1);
				bottom=getmap(xnew,seg->y-1);
				}
			else {
				top = getmap(seg->x,seg->y-1);
				bottom = getmap(seg->x,seg->y);
				}
			if( top==NULL_PLATE && bottom == NULL_PLATE ) {
				fprintf( stderr, "error: bad segment!\n");
				exit(-1);
				}
			else if( top==bottom ) {
```

```
                if( seg->left != NULL ) seg->left->right = seg->right;
                if( seg->right != NULL ) seg->right->left = seg->left;
                delete_segment( seg, plate );
                if( plate->active == NULL ) return(TRUE); else return(FALSE);
                }
        else if( top==NULL_PLATE ) {
                if( seg->y == 1 ) {
                        setmap(xnew,seg->y,bottom);
                        s1=insert_segment( plate, HORIZONTAL, xnew,          (seg->y+1) );
                        s2=insert_segment( plate, VERTICAL,   wrap(xnew+1),  (seg->y)   );
                        s3=insert_segment( plate, VERTICAL,   xnew,          (seg->y)   );
                        }
                else {
                        setmap(seg->x,seg->y-1,bottom);
                        s1=insert_segment( plate, HORIZONTAL, wrap(seg->x),   (seg->y-1)   );
                        s2=insert_segment( plate, VERTICAL,   wrap(seg->x),   (seg->y-1)   );
                        s3=insert_segment( plate, VERTICAL,   wrap(seg->x+1), (seg->y-1)   );
                        }
                linkup( seg->left, s2, s1, s3, seg->right );
                delete_segment( seg, plate );
                return(TRUE);
                }
        else if( bottom==NULL_PLATE ) {
                if( seg->y == (N_ROWS-1) ) {
                        setmap(xnew,seg->y-1,top);
                        s1=insert_segment( plate, HORIZONTAL, xnew,          (seg->y-1) );
                        s2=insert_segment( plate, VERTICAL,   wrap(xnew+1),  (seg->y-1) );
                        s3=insert_segment( plate, VERTICAL,   xnew,          (seg->y-1) );
                        }
                else {
                        setmap(seg->x,seg->y,top);
                        s1=insert_segment( plate, HORIZONTAL, wrap(seg->x),   (seg->y+1)   );
                        s2=insert_segment( plate, VERTICAL,   wrap(seg->x),   (seg->y)   );
                        s3=insert_segment( plate, VERTICAL,   wrap(seg->x+1), (seg->y)   );
                        }
                linkup( seg->left, s3, s1, s2, seg->right );
                delete_segment( seg, plate );
                return(TRUE);
                }
        else /* different plates */ {
                move_segment( seg, plate );
                if( plate->active == NULL ) return(TRUE); else return(FALSE);
                }
        }
else /* seg->orientation == VERTICAL */ {
        left =  getmap(seg->x-1,seg->y);
        right = getmap(seg->x,seg->y);
```

```
		if( left==NULL_PLATE && right == NULL_PLATE ) {
			fprintf( stderr, "error: bad segment!\n");
			exit(-1);
			}
		else if( left==right ) {
			if( seg->left != NULL ) seg->left->right = seg->right;
			if( seg->right != NULL ) seg->right->left = seg->left;
			delete_segment( seg, plate );
			if( plate->active == NULL ) return(TRUE); else return(FALSE);
			}
		else if( left==NULL_PLATE ) {
			setmap(seg->x-1,seg->y,right);
			s1=insert_segment( plate, VERTICAL,     wrap(seg->x-1),   (seg->y)   );
			s2=insert_segment( plate, HORIZONTAL,   wrap(seg->x-1),   (seg->y)   );
			s3=insert_segment( plate, HORIZONTAL,   wrap(seg->x-1),   (seg->y+1) );
			linkup( seg->left, s3, s1, s2, seg->right );
			delete_segment( seg, plate );
			return(TRUE);
			}
		else if( right==NULL_PLATE ) {
			setmap(seg->x,seg->y,left);
			s1=insert_segment( plate, VERTICAL,     wrap(seg->x+1),   (seg->y)   );
			s2=insert_segment( plate, HORIZONTAL,   wrap(seg->x),     (seg->y)   );
			s3=insert_segment( plate, HORIZONTAL,   wrap(seg->x),     (seg->y+1) );
			linkup( seg->left, s2, s1, s3, seg->right );
			delete_segment( seg, plate );
			return(TRUE);
			}
		else /* different plates */ {
			move_segment( seg, plate );
			if( plate->active == NULL ) return(TRUE); else return(FALSE);
			}
		}
	}

delete_segment( seg, plate )
struct segment *seg;
struct plate *plate;
{
	plate->active_count --;

	if( plate->active == seg ) { /*head of list*/
		if( seg->next != NULL ) seg->next->last=NULL;
		plate->active = seg->next;
		}
	else if( seg->next == NULL ) { /*tail of list */
		if( seg->last != NULL ) seg->last->next = NULL;
```

```
                }
        else { /* interior of list */
                seg->last->next = seg->next;
                seg->next->last = seg->last;
                }

        if( plate->active_count == 0 ) {
                active_plates--;
                set_lookup();
                }

        free(seg);
        }

struct segment *insert_segment( plate, type, x, y )
struct plate *plate;
int type, x, y;
{
        int top, bottom, left, right;
        int xnew, x1, x2, x3, y1, y2, y3;
        struct segment *seg;
        if( type == VERTICAL ) {
                left = getmap(x-1,y);
                right = getmap(x,y);
                if( left==NULL_PLATE && right==NULL_PLATE ) {
                        fprintf(stderr,"error: attempt to insert segment between null pixels!\n");
                        exit(-1);
                        }
                else if( left==right ) { /*ignore this insert; its interior to a plate*/
                        seg=NULL;
                        }
                else if( left==NULL_PLATE || right==NULL_PLATE ) { /*insert on active list*/
                        seg = (struct segment *) malloc( sizeof(struct segment) );
                        if( seg == NULL ) {
                                fprintf(stderr,"error: out of memory!\n");
                                exit(-1);
                                }
                        seg->last = NULL;
                        seg->next = plate->active;
                        seg->x=x;
                        seg->y=y;
                        seg->orientation=type;
                        if( plate->active != NULL ) plate->active->last = seg;
                        plate->active = seg;
                        plate->active_count++;
                        }
                else /* insert on inactive list */ {
```

```
                        seg = (struct segment *) malloc( sizeof(struct segment) );
                        if( seg == NULL ) {
                                fprintf(stderr,"error: out of memory!\n");
                                exit(-1);
                                }
                        seg->last = NULL;
                        seg->next = plate->inactive;
                        seg->x=x;
                        seg->y=y;
                        seg->orientation=type;
                        if( plate->inactive != NULL ) plate->inactive->last = seg;
                        plate->inactive = seg;
                        plate->inactive_count++;
                        }
                }
        else /* type == HORIZONTAL */ {
                if( y==1 ){ /*top of map*/
                        xnew=slide(x);
                        top=getmap(xnew,1);
                        bottom=getmap(x,y);
                        }
                else if( y == (N_ROWS-1) ) {
                        xnew=slide(x);
                        top=getmap(x,y-1);
                        bottom=getmap(xnew,y-1);
                        }
                else {
                        top = getmap(x,y-1);
                        bottom = getmap(x,y);
                        }
                if( top==NULL_PLATE && bottom==NULL_PLATE ) {
                        fprintf(stderr,"error: attempt to insert segment between null pixels!\n");
                        exit(-1);
                        }
                else if( top==bottom ) { /*ignore this insert; its interior to a plate*/
                        seg=NULL;
                        }
                else if( top==NULL_PLATE || bottom==NULL_PLATE ) { /*insert on active list*/
                        seg = (struct segment *) malloc( sizeof(struct segment) );
                        if( seg == NULL ) {
                                fprintf(stderr,"error: out of memory!\n");
                                exit(-1);
                                }
                        seg->last = NULL;
                        seg->next = plate->active;
                        seg->x=x;
                        seg->y=y;
```

```
                        seg->orientation=type;
                        if( plate->active != NULL ) plate->active->last = seg;
                        plate->active = seg;
                        plate->active_count++;
                        }
                else /* insert on inactive list */ {
                        seg = (struct segment *) malloc( sizeof(struct segment) );
                        if( seg == NULL ) {
                                fprintf(stderr,"error: out of memory!\n");
                                exit(-1);
                                }
                        seg->last = NULL;
                        seg->next = plate->inactive;
                        seg->x=x;
                        seg->y=y;
                        seg->orientation=type;
                        if( plate->inactive != NULL ) plate->inactive->last = seg;
                        plate->inactive = seg;
                        plate->inactive_count++;
                        }
                }
        return(seg);
        }

move_segment( seg, plate )
struct segment *seg;
struct plate *plate;
{
        /* unlink from active list */
        plate->active_count --;

        if( plate->active == seg ) { /*head of list*/
                if( seg->next != NULL ) seg->next->last=NULL;
                plate->active = seg->next;
                }
        else if( seg->next == NULL ) { /*tail of list */
                if( seg->last != NULL ) seg->last->next = NULL;
                }
        else { /* interior of list */
                seg->last->next = seg->next;
                seg->next->last = seg->last;
                }

        /* link onto inactive list */
        seg->last=NULL;
        seg->next=plate->inactive;
        if( plate->inactive != NULL ) plate->inactive->last=seg;
```

```
        plate->inactive=seg;
        plate->inactive_count++;

        if( plate->active_count == 0 ) {
                active_plates--;
                set_lookup();
                }

        }

traverse( head )
struct segment *head;
{
        struct segment *current;
        int i;

        current=head;
        i=0;
        while( current!=NULL ) {
                printf("%d: %d %d %d %d %d\n", i, current->orientation, current->x, current->y, current->last, current->next
                i++;
                current=current->next;
                }

        }

traverse2( head )
struct segment *head;
{
        struct segment *current;
        int i;

        current=head;
        i=0;
        while( current!=NULL ) {
                printf("%d: %d %d %d %d %d\n", i, current->orientation, current->x, current->y, current->last, current->next
                i++;
                current=current->right;
                if( current==head ) break;
                }

        }

set_lookup()
{
        int i, j, k, n, count;
```

```
        double current_percentage;

        count=0;
        current_percentage = 0.0;
        for( i=0; i<N_PLATES; i++ ) {
                if( the_plate[i].active != NULL ) {
                        count++;
                        current_percentage += percentage[i];
                        }
                }

        for( i=0; i<10000; i++ ) lookup[i]=PHONEY_PLATE;

        k=0;
        for( i=0; i<N_PLATES; i++ ) {
                if( the_plate[i].active != NULL ) {
                        n = ((int)(100.0 * percentage[i] * original_total_percentage / current_percentage+0.5));
                        for( j=0; j<n; j++ ) {
                                if( k>=10000 ) break;
                                lookup[k]=i;
                                k++;
                                }
                        }
                }
        }


cleanup_segments()
{
        int i, top, bottom, left, right, xnew;
        struct segment *seg, *next;
        struct plate *plate;

        for( i=0; i<N_PLATES; i++ ) {
                plate = &(the_plate[i]);
                seg = plate->active;          inactive
                while( seg != NULL ) {
                        next=seg->next;
                        if( seg->orientation == HORIZONTAL ) {
                                if( seg->y==1 ){ /*top of map*/
                                        xnew=slide(seg->x);
                                        top=getmap(xnew,1);
                                        bottom=getmap(seg->x,seg->y);
                                        }
                                else if( seg->y == (N_ROWS-1) ) {
                                        xnew=slide(seg->x);
                                        top=getmap(seg->x,seg->y-1);
```

```
                        bottom=getmap(xnew,seg->y-1);
                        }
                else {
                        top = getmap(seg->x,seg->y-1);
                        bottom = getmap(seg->x,seg->y);
                        }
                if( top==NULL_PLATE && bottom == NULL_PLATE ) {
                        fprintf( stderr, "error: bad segment!\n");
                        exit(-1);
                        }
                else if( top==bottom ) {
                        if( seg->left != NULL ) seg->left->right = seg->right;
                        if( seg->right != NULL ) seg->right->left = seg->left;
                        delete_segment( seg, plate );
                        }
                else if( top==NULL_PLATE ) {
                        ;
                        }
                else if( bottom==NULL_PLATE ) {
                        ;
                        }
                else /* different plates */ {
                        move_segment( seg, plate );
                        }
                }
        else /* seg->orientation == VERTICAL */ {
                left =  getmap(seg->x-1,seg->y);
                right = getmap(seg->x,seg->y);
                if( left==NULL_PLATE && right == NULL_PLATE ) {
                        fprintf( stderr, "error: bad segment!\n");
                        exit(-1);
                        }
                else if( left==right ) {
                        if( seg->left != NULL ) seg->left->right = seg->right;
                        if( seg->right != NULL ) seg->right->left = seg->left;
                        delete_segment( seg, plate );
                        }
                else if( left==NULL_PLATE ) {
                        ;
                        }
                else if( right==NULL_PLATE ) {
                        ;
                        }
                else /* different plates */ {
                        move_segment( seg, plate );
                        }
                }
```

```c
                        seg = next;
                        } /*end while segment */
                } /*end for plates */
        }

linkup( s1, s2, s3, s4, s5 )
struct segment *s1, *s2, *s3, *s4, *s5;
{
        struct segment *s[3];
        int i, n;

        n=0;
        if( s2 != NULL ) s[n++]=s2;
        if( s3 != NULL ) s[n++]=s3;
        if( s4 != NULL ) s[n++]=s4;

        if( n==3 ) {
                if( s1 != NULL ) s1->right = s[0];
                s[0]->left = s1;   s[0]->right=s[1];
                s[1]->left = s[0]; s[1]->right=s[2];
                s[2]->left = s[1]; s[2]->right=s5;
                if( s5 != NULL ) s5->left = s[2];
                }
        else if( n==2 ) {
                if( s1 != NULL ) s1->right = s[0];
                s[0]->left = s1;   s[0]->right=s[1];
                s[1]->left = s[0]; s[1]->right=s5;
                if( s5 != NULL ) s5->left = s[1];
                }
        else if( n==1 ) {
                if( s1 != NULL ) s1->right = s[0];
                s[0]->left = s1; s[0]->right=s5;
                if( s5 != NULL ) s5->left = s[0];
                }
        else if( n==0 ) {
                if( s1 != NULL ) s1->right = s5;
                if( s5 != NULL ) s5->left = s1;
                }

        }

do_slide(x)
int x;
{
        return(slide(x));
        }
```

```
free_up_list(head)
struct segment *head;
{
        struct segment *current, *next;
        current=head;
        while( current!=NULL ) {
                next = current->next;
                free(current);
                current=next;
                }
        }
```