

1. Introduction. This is a hastily written implementation of hull insertion.

```

format Graph int    /* gb_graph defines the Graph type and a few others */
format Vertex int
format Arc int
format Area int
#include "gb_graph.h"
#include "gb_miles.h"
int n = 128;
⟨Global variables 2⟩
⟨Procedures 11⟩
main()
{
    ⟨Local variables 6⟩
    Graph *g = miles(128, 0, 0, 0, 0, 0, 0);
    mems = ccs = 0;
    ⟨Find convex hull of g 7⟩;
    printf("Total of %d mems and %d calls on ccw.\n", mems, ccs);
}

```

2. I'm instrumenting this in a simple way.

```

#define o mems++
#define oo mems += 2
⟨Global variables 2⟩ ≡
    int mems;    /* memory accesses */
    int ccs;    /* calls on ccw */
    int serial_no = 1; /* used to disambiguate entries with equal coordinates */

```

See also section 4.

This code is used in section 1.

3. Data structures. For now, each vertex is represented by two coordinates stored in the utility fields $x.I$ and $y.I$. I'm also putting a serial number into $z.I$, so that I can check whether different algorithms generate identical hulls.

A vertex v in the convex hull also has a successor $v\text{-succ}$ and a predecessor $v\text{-pred}$, stored in utility fields u and v .

This implementation is the simplest one I know; it simply walks around the current convex hull each time, therefore not really bad if the current hull never gets big.

```
#define succ u.V
```

```
#define pred v.V
```

4. \langle Global variables 2 $\rangle \equiv$

```
Vertex *rover; /* one of the vertices in the convex hull */
```

5. We assume that the vertices have been given to us in a GraphBase-type graph. The algorithm begins with a trivial hull that contains only the first two vertices.

\langle Initialize the data structures 5 $\rangle \equiv$

```
o, u = g-vertices;
```

```
v = u + 1;
```

```
u-z.I = 0;
```

```
v-z.I = 1;
```

```
oo, u-succ = u-pred = v;
```

```
oo, v-succ = v-pred = u;
```

```
rover = u;
```

```
if (n < 150) printf("Beginning with %s; %s\n", u-name, v-name);
```

This code is used in section 7.

6. We'll probably need a bunch of local variables to do elementary operations on data structures.

\langle Local variables 6 $\rangle \equiv$

```
Vertex *u, *v, *vv, *w;
```

This code is used in section 1.

7. Hull updating. The main loop of the algorithm updates the data structure incrementally by adding one new vertex at a time. If the new vertex lies outside the current convex hull, we put it into the cycle and possibly delete some vertices that were previously part of the hull.

```

⟨ Find convex hull of g 7 ⟩ ≡
  ⟨ Initialize the data structures 5 ⟩;
  for (oo, vv = g-vertices + 2; vv < g-vertices + g-n; vv++) {
    vv→z.I = ++serial_no;
    ⟨ Go around the current hull; continue if vv is inside it 9 ⟩;
    ⟨ Update the convex hull, knowing that vv lies outside the consecutive hull vertices u and v 10 ⟩;
  }
  ⟨ Print the convex hull 8 ⟩;

```

This code is used in section 1.

8. Let me do the easy part first, since it's bedtime and I can worry about the rest tomorrow.

```

⟨ Print the convex hull 8 ⟩ ≡
  u = rover;
  printf("The convex hull is:\n");
  do {
    printf("%s\n", u→name);
    u = u→succ;
  } while (u ≠ rover);

```

This code is used in section 7.

9. ⟨ Go around the current hull; **continue** if *vv* is inside it 9 ⟩ ≡

```

  u = rover;
  do {
    o, v = u→succ;
    if (ccw(u, vv, v)) goto found;
    u = v;
  } while (u ≠ rover);
  continue;
found: ;

```

This code is used in section 7.

10. \langle Update the convex hull, knowing that vv lies outside the consecutive hull vertices u and v 10 $\rangle \equiv$

```

if ( $u \equiv rover$ ) {
  while (1) {
     $o, w = u \rightarrow pred$ ;
    if ( $w \equiv v$ ) break;
    if ( $ccw(vv, w, u)$ ) break;
     $u = w$ ;
  }
   $rover = w$ ;
}
while (1) {
  if ( $v \equiv rover$ ) break;
   $o, w = v \rightarrow succ$ ;
  if ( $ccw(w, vv, v)$ ) break;
   $v = w$ ;
}
 $oo, u \rightarrow succ = v \rightarrow pred = vv$ ;
 $oo, vv \rightarrow pred = u$ ;  $vv \rightarrow succ = v$ ;
if ( $n < 150$ )  $printf("New\_hull\_sequence\_(\%s; \_\%s; \_\%s) \backslash n", u \rightarrow name, vv \rightarrow name, v \rightarrow name)$ ;

```

This code is used in section 7.

11. Determinants. I need code for the primitive function *ccw*. Floating-point arithmetic suffices for my purposes.

We want to evaluate the determinant

$$ccw(u, v, w) = \begin{vmatrix} u(x) & u(y) & 1 \\ v(x) & v(y) & 1 \\ w(x) & w(y) & 1 \end{vmatrix} = \begin{vmatrix} u(x) - w(x) & u(y) - w(y) \\ v(x) - w(x) & v(y) - w(y) \end{vmatrix}.$$

```

⟨Procedures 11⟩ ≡
int ccw(u, v, w)
    Vertex *u, *v, *w;
    { register double wx = (double) w→x.I, wy = (double) w→y.I;
      register double det = ((double) u→x.I - wx) * ((double) v→y.I - wy) - ((double)
        u→y.I - wy) * ((double) v→x.I - wx);
      Vertex *uu = u, *vv = v, *ww = w, *t;
      if (det ≡ 0) {
        det = 1;
        if (u→x.I > v→x.I ∨ (u→x.I ≡ v→x.I ∧ (u→y.I > v→y.I ∨ (u→y.I ≡ v→y.I ∧ u→z.I > v→z.I)))) {
          t = u; u = v; v = t; det = -det;
        }
        if (v→x.I > w→x.I ∨ (v→x.I ≡ w→x.I ∧ (v→y.I > w→y.I ∨ (v→y.I ≡ w→y.I ∧ v→z.I > w→z.I)))) {
          t = v; v = w; w = t; det = -det;
        }
        if (u→x.I > v→x.I ∨ (u→x.I ≡ v→x.I ∧ (u→y.I > v→y.I ∨ (u→y.I ≡ v→y.I ∧ u→z.I < v→z.I)))) {
          det = -det;
        }
      }
      if (n < 150)
        printf("cc(%s; %s; %s) is %s\n", uu→name, vv→name, ww→name, det > 0 ? "true" : "false");
        ccs++;
      return (det > 0);
    }

```

This code is used in section 1.

ccs: 1, 2, 11.

ccw: 2, 9, 10, 11.

det: 11.

found: 9.

g: 1.

gb_graph: 1.

Graph: 1.

main: 1.

mems: 1, 2.

miles: 1.

n: 1.

name: 5, 8, 10, 11.

o: 2.

oo: 2, 5, 7, 10.

pred: 3, 5, 10.

printf: 1, 5, 8, 10, 11.

rover: 4, 5, 8, 9, 10.

serial_no: 2, 7.

succ: 3, 5, 8, 9, 10.

t: 11.

u: 6, 11.

uu: 11.

v: 6, 11.

Vertex: 4, 6, 11.

vertices: 5, 7.

vv: 6, 7, 9, 10, 11.

w: 6, 11.

ww: 11.

wx: 11.

wy: 11.

- ⟨ Find convex hull of g 7 ⟩ Used in section 1.
- ⟨ Global variables 2, 4 ⟩ Used in section 1.
- ⟨ Go around the current hull; **continue** if vv is inside it 9 ⟩ Used in section 7.
- ⟨ Initialize the data structures 5 ⟩ Used in section 7.
- ⟨ Local variables 6 ⟩ Used in section 1.
- ⟨ Print the convex hull 8 ⟩ Used in section 7.
- ⟨ Procedures 11 ⟩ Used in section 1.
- ⟨ Update the convex hull, knowing that vv lies outside the consecutive hull vertices u and v 10 ⟩ Used in section 7.