1.   Introduction.   This is a hastily written implementation of the daghull algorithm.

> **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 13 ⟩
> *main* ( )
> {
>   ⟨ Local variables 7 ⟩
>   **Graph** $*g = miles(128, 0, 0, 0, 0, 0, 0)$;
> $mems = ccs = 0$;
>   ⟨ Find convex hull of g 8 ⟩;
>   *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* ∗/

See also section 5.

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 $v\rightarrow succ$ and and predecessor $v\rightarrow pred$, stored in utility fields u and v. There's also a pointer to an "instruction," $v\rightarrow inst$, for which I'm using an arc record although I need only two fields; that one goes in utility field w.

An instruction has two parts. Its *tip* is a vertex to be used in counterclockwise testing. Its *next* is another instruction to be followed; or, if Λ, there's no next instruction; or, if a pointer to the smallest possible instruction, we will do something special to update the current convex hull.

#**define** *succ* $u.V$
#**define** *pred* $v.V$
#**define** *inst* $w.A$

4. ⟨Initialize the array of instructions 4⟩ ≡
  $\textit{first\_inst} = (\mathbf{Arc} \; *) \; \textit{gb\_alloc}((4 * g\rightarrow n - 2) * \mathbf{sizeof}(\mathbf{Arc}), \textit{working\_storage});$
  if $(\textit{first\_inst} \equiv \Lambda)$ **return** (1);   /∗ fixthis ∗/
  $\textit{next\_inst} = \textit{first\_inst};$
This code is used in section 6.

5. ⟨Global variables 2⟩ +≡
  Arc ∗*first_inst*;   /∗ beginning of the array of instructions ∗/
  Arc ∗*next_inst*;   /∗ first unused slot in that array ∗/
  Vertex ∗*rover*;   /∗ one of the vertices in the convex hull ∗/
  Area *working_storage*;
  int $\textit{serial\_no} = 1;$ /∗ used to disambiguate entries with equal coordinates ∗/

6. 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.
⟨Initialize the data structures 6⟩ ≡
  $\textit{init\_area}(\textit{working\_storage});$
  ⟨Initialize the array of instructions 4⟩;
  $o, u = g\rightarrow vertices;$
  $v = u + 1;$
  $u\rightarrow z.I = 0;$
  $v\rightarrow z.I = 1;$
  $oo, u\rightarrow succ = u\rightarrow pred = v;$
  $oo, v\rightarrow succ = v\rightarrow pred = u;$
  $oo, \textit{first\_inst}\rightarrow tip = u; \; \textit{first\_inst}\rightarrow next = \textit{first\_inst};$
  $oo, (\mathord{+}\mathord{+}\textit{next\_inst})\rightarrow tip = v; \; \textit{next\_inst}\rightarrow next = \textit{first\_inst};$
  $o, u\rightarrow inst = \textit{first\_inst};$
  $o, v\rightarrow inst = \textit{next\_inst}\mathord{+}\mathord{+};$
  $\textit{rover} = u;$
  if $(n < 150)$ $\textit{printf}(\texttt{"Beginning\textvisiblespace with\textvisiblespace(\%s;\textvisiblespace\%s)\textbackslash n"}, u\rightarrow name, v\rightarrow name);$
This code is used in section 8.

7. We'll probably need a bunch of local variables to do elementary operations on data structures.
⟨Local variables 7⟩ ≡
  Vertex ∗u, ∗v, ∗vv, ∗w;
  Arc ∗p, ∗q, ∗r, ∗s;
This code is used in section 1.

8.   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 8 ⟩ ≡
  ⟨ Initialize the data structures 6 ⟩;
  for (oo, $vv = g$→$vertices + 2$; $vv < g$→$vertices + g$→n; vv++) {
    vv→$z.I = ++serial\_no$;
    ⟨ Follow the instructions; **continue** if vv is inside the current hull 10 ⟩;
    ⟨ Update the convex hull, knowing that vv lies outside the consecutive hull vertices u and v 11 ⟩;
  }
  ⟨ Print the convex hull 9 ⟩;
This code is used in section 1.

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

⟨ Print the convex hull 9 ⟩ ≡
  $u = rover$;
  $printf($ "The␣convex␣hull␣is:\n" $)$;
  do {
    $printf($ "␣␣%s\n", u→$name$ $)$;
    $u = u$→$succ$;
  } **while** ($u \neq rover$);
This code is used in section 8.

10.   ⟨ Follow the instructions; **continue** if vv is inside the current hull 10 ⟩ ≡
  $p = first\_inst$;
  do {
    if (oo, $ccw(p$→$tip, vv, (p + 1)$→$tip)$) p++;
    $q = p$;
    $o, p = p$→$next$;
  } **while** ($p > first\_inst$);
  if ($p \equiv \Lambda$) **continue**;
  $o, v = q$→tip;
  $o, u = v$→$pred$;
This code is used in section 8.

11. ⟨Update the convex hull, knowing that *vv* lies outside the consecutive hull vertices *u* and *v* 11⟩ ≡

   o, *q*→*next* = *next_inst*;

   while (1) {

      o, *w* = *u*→*pred*;

      if (*w* ≡ *v*) **break**;

      if (*ccw*(*vv*, *w*, *u*)) **break**;

      o, *u*→*inst*→*next* = *next_inst*;

      if (*rover* ≡ *u*) *rover* = *w*;

      u = *w*;

   }

   while (1) {

      o, *w* = *v*→*succ*;

      if (*w* ≡ *u*) **break**;

      if (*ccw*(*w*, *vv*, *v*)) **break**;

      if (*rover* ≡ *v*) *rover* = *w*;

      v = *w*;

      o, *v*→*inst*→*next* = *next_inst*;

   }

   oo, *u*→*succ* = *v*→*pred* = *vv*;

   oo, *vv*→*pred* = *u*; *vv*→*succ* = *v*;

   ⟨Compile two new instructions, for (*u*, *vv*) and (*vv*, *v*) 12⟩;

This code is used in section 8.

12. ⟨Compile two new instructions, for (*u*, *vv*) and (*vv*, *v*) 12⟩ ≡

   o, *next_inst*→*tip* = *vv*;

   o, *next_inst*→*next* = *first_inst*;

   o, *vv*→*inst* = *next_inst*;

   next_inst ++;

   o, *next_inst*→*tip* = *u*;

   o, *next_inst*→*next* = *next_inst* + 1;

   next_inst ++;

   o, *next_inst*→*tip* = *v*;

   o, *next_inst*→*next* = *first_inst*;

   o, *v*→*inst* = *next_inst*;

   next_inst ++;

   o, *next_inst*→*tip* = *vv*;

   o, *next_inst*→*next* = Λ;

   next_inst ++;

   if (*n* < 150) *printf*("New␣hull␣sequence␣(%s;␣%s;␣%s)\n", *u*→*name*, *vv*→*name*, *v*→*name*);

This code is used in section 11.

13. 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 13 ⟩ ≡
```
  int  ccw (u, v, w)
      Vertex *u, *v, *w;
  { register double wx = (double) w⃗x.I, wy = (double) w⃗y.I;
    register double det = ((double) u⃗x.I − wx) ∗ ((double) v⃗y.I − wy) − ((double)
        u⃗y.I − wy) ∗ ((double) v⃗x.I − wx);
    Vertex *uu = u, *vv = v, *ww = w, *t;
    if (det ≡ 0) {
      det = 1;
      if (u⃗x.I > v⃗x.I ∨ (u⃗x.I ≡ v⃗x.I ∧ (u⃗y.I > v⃗y.I ∨ (u⃗y.I ≡ v⃗y.I ∧ u⃗z.I > v⃗z.I)))) {
        t = u; u = v; v = t; det = −det;
      }
      if (v⃗x.I > w⃗x.I ∨ (v⃗x.I ≡ w⃗x.I ∧ (v⃗y.I > w⃗y.I ∨ (v⃗y.I ≡ w⃗y.I ∧ v⃗z.I > w⃗z.I)))) {
        t = v; v = w; w = t; det = −det;
      }
      if (u⃗x.I > v⃗x.I ∨ (u⃗x.I ≡ v⃗x.I ∧ (u⃗y.I > v⃗y.I ∨ (u⃗y.I ≡ v⃗y.I ∧ u⃗z.I < v⃗z.I)))) {
        det = −det;
      }
    }
    if (n < 150)
      printf ("cc(%s;␣%s;␣%s)␣is␣%s\n", uu⃗name, vv⃗name, ww⃗name, det > 0 ? "true" : "false");
    ccs ++;
    return (det > 0);
  }
```
This code is used in section 1.

Arc: 4, 5, 7.	*next_inst*: 4, 5, 6, 11, 12.
Area: 5.	*o*: 2.
ccs: 1, 2, 13.	*oo*: 2, 6, 8, 10, 11.
ccw: 2, 10, 11, 13.	*p*: 7.
det: 13.	*pred*: 3, 6, 10, 11.
first_inst: 4, 5, 6, 10, 12.	*printf*: 1, 6, 9, 12, 13.
g: 1.	*q*: 7.
gb_alloc: 4.	*r*: 7.
gb_graph: 1.	*rover*: 5, 6, 9, 11.
Graph: 1.	*s*: 7.
init_area: 6.	*serial_no*: 5, 8.
inst: 3, 6, 11, 12.	*succ*: 3, 6, 9, 11.
main: 1.	*t*: 13.
mems: 1, 2.	*tip*: 3, 6, 10, 12.
miles: 1.	*u*: 7, 13.
n: 1.	*uu*: 13.
name: 6, 9, 12, 13.	*v*: 7, 13.
next: 3, 6, 10, 11, 12.	**Vertex**: 5, 7, 13.