Constructing the layer representation of polygons in parallel

F. Martínez, A.J. Rueda and F.R. Feito
Depto. de Informática (Universidad de Jaén)
Campus Las Lagunillas
23071, Jaén, Spain
{fmartin,ajrueda,ffeito}@ujaen.es

ABSTRACT

For several years we have developed a scheme for representing polygons and 3D polyhedra by means of layers of triangles and tetrahedra, together with effective algorithms for basic geometric operations on the represented objects. In this paper we explore the optimization of some of these algorithms by parallel processing techniques.

Keywords
Graphic object representations, parallel geometric algorithms

1. INTRODUCTION

In previous work [Rue02a, Fei99a] we have developed a method for representing polygons based on layers of triangles, showing several interesting properties: it is valid for any kind of polygon, simple, has little space needs, and makes easy the effective implementation of several operations, as the point-in-polygon inclusion test [Fei95a, Fei97a] or Boolean operations on polygons [Riv00a].

During the development of our work we have had some problems with the algorithms used for constructing the polygon representation. The most efficient, based on a radial line sweeping, needs $O(nlogn)$ time, where *n* is the number of vertices of the polygon. However, this algorithm is not extensible to 3D, so that we are more interested in a simpler, 3D extensible algorithm that requires $O(n^2)$ time and space.

Realistic scenes have a great number of polygons containing a great number of vertices. So, in order to speed up the representation of scenes we must optimize the algorithm used for constructing polygons.

Conference proceedings ISBN 80-903100-9-5
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.

For this reason we have decided to parallelize the $O(n^2)$ constructing algorithm, and this experience is described in the rest of the paper.

The remainder of the paper is structured as follows. Section 2 briefly describes the representation of polygons by layers. Section 3 explains the sequential, $O(n^2)$ algorithm used for constructing the representation. Section 4 shows how to parallelize this algorithm. Section 5 compares execution times of both algorithms. Finally, Section 6 brings conclusions and states the future directions of our research.

2. THE LAYER REPRESENTATION OF POLYGONS

This section briefly describes the representation of polygons by layers, so that the algorithm used for its construction, explained in the next section, can be understood. See [Rue02a, Fei99a] for a detailed description of the contents of this section.

Given a polygon, it is possible to define a set of triangles between an arbitrary point, named *origin*, and each edge of the polygon. These triangles are called *origin triangles* because one of their vertices coincides with the origin. An origin triangle has two *origin edges* whose vertices are determined by the origin and a vertex of the polygon, and a *non-origin edge* which belongs to the polygon. Figure 1 shows a polygon and their corresponding origin triangles.

The layer representation is based on the concepts of: *subordination relation*, *subordination chain*, *index*, and *layer*.

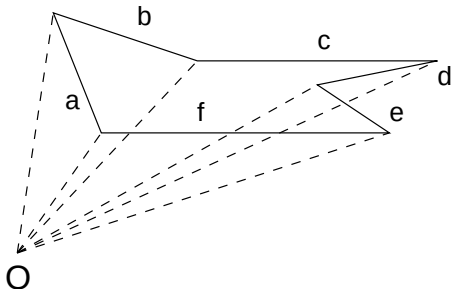

Figure 1: A polygon and its origin triangles.

Definition 1. Let *t* and *s* be two origin triangles of a polygon. We say that *t* is subordinate to *s* ($t \triangleleft s$) if and only if $t = s$ or, in the case that $t \neq s$, exists a point belonging to the unidimensional interior of the non-origin edge of *t* that also belongs to the interior of the triangle *s*. In Figure 1 *a* is subordinate to *b* ($a \triangleleft b$) because the interior of *b* clearly contains a big portion of the non-origin edge of *a*. Triangles *b* and *c* are not subordinate to any other one.

Definition 2. A *subordination chain* is defined as an ordered sequence of triangles $S = \{s_0 s_1 \dots s_n\}$ where $s_i \triangleleft s_{i+1} \forall i : 0 \leq i < n$. In Figure 1 all non-trivial subordination chains are $\{a, b\}$, $\{f, e\}$, $\{f, e, d\}$, $\{f, e, d, c\}$, $\{f, b\}$, $\{f, c\}$, $\{f, d\}$, $\{e, d\}$, $\{e, d, c\}$, $\{e, c\}$ and $\{d, c\}$.

Definition 3. We define the *index* of an origin triangle *s* ($ind(s)$) as the length of the longest subordination chain of the set of ordered triangles determined by a polygon *P*, starting with triangle *s*. Formally, $ind(s) = max\{|S_i|\}$ where $S_i = \{s_0 s_1 \dots s_n\}$ is any valid subordination chain as defined previously verifying $s = s_0$. In Figure 1 $ind(a) = 2$, $ind(b) = 1$, $ind(c) = 1$, $ind(d) = 2$, $ind(e) = 3$ and $ind(f) = 4$

Triangles can be sorted in several *layers* according to their index. This is the *layer representation* or *L-REP* of a polygon, as we see in the next definition.

Definition 4. We define the *layer representation* or *L-REP* of a polygon *P* as the set of layers containing the triangles with the same index, that is $R = \{L_1 L_2 \dots L_l\}$ where $L_i = \{s_j : ind(s_j) = i\}$ is the layer of origin triangles s_j with index *i*, and *l* is the maximum index of the origin triangles generated by the polygon *P*.

Figure 2 illustrates the representation of a polygon by its L-REP.

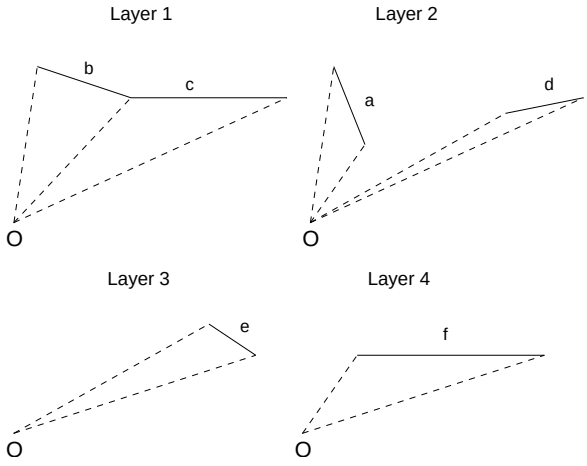

Figure 2: A polygon and its L-REP.

3. SEQUENTIAL CONSTRUCTION OF THE L-REP

In this section we describe the sequential algorithm used for constructing the layer representation of a polygon. Its pseudocode is shown below.

```
int n = p.n_edges(); // # edges
bool M[n][n];      // sub. matrix
int sc[n];         // sub. counters

// Stage 1: Build M and sc
for (int i = 0; i < n; i++) {
  sc[i] = 0;
  for (int j = 0; j < n; j++) {
    M[i][j] = p.edge(i).sub_to
              (p.edge(j));
    sc[i] += M[i][j] ? 1 : 0;
  }
}

// Stage 2: Build layers
int ntp = n;  // # triangles processed
int l = 0;    // current layer
vector<int> sct(sc); // copy of sc
while (ntp > 0) {
  l++;
  p.lrep.add_layer(l);
  for (int i = 0; i < n; i++) {
    if (sc[i] == 0) {
      p.lrep.add_triangle(l, p.edge(i));
      for (int j = 0; j < n; j++) {
        if (M[j][i]) {
          sct[j]--;
        }
      }
      sct[i] = -1;  // inserted
      ntp--;
    }
  }
  sc = sct;
}
```

The L-REP of a polygon *p* with *n* edges is constructed in two stages. In the first stage two data structures are built: the subordination matrix (*M*) and the subordination counters (*sc*). *M* is a nxn boolean matrix that stores at position (i, j) if $s_i \triangleleft s_j$, where s_x refers to the origin triangle associated to the edge *x* of *p*. The *sc* vector stores at position *i* the number of triangles to which s_i is subordinate — i.e., the origin triangle associated to the edge *i* of *p*.

From the data structures built in the first stage, *M* and *sc*, is easy to calculate the layers of a polygon taking into account the following remark: after stage 1, triangles whose subordination counter are 0 belong to layer 1. If, for every layer 1 triangle s_i, the subordination counters of the triangles s_j such that $s_j \triangleleft s_i$ are decremented by 1, the new triangles whose subordination counter are 0 belong to layer 2. The same reasoning can be applied to compute the layer 3 triangles from layer 2 triangles, and so on, all the layers of a polygon.

4. PARALLEL CONSTRUCTION OF THE L-REP

We will apply the *data domain decomposition* scheme to parallelize the sequential algorithm described in the previous section. Following this scheme, a data domain to be processed is divided into several disjoint "chunks", and every "chunk" is allocated to a different process that runs an algorithm on it. If, as in our case, all processes run the same algorithm we have a SPMD (*Single Program Multiple Data*) application.

In our parallelization the data domain consists of the polygon edges, — i.e., their associated origin triangles. A different subset of origin triangles is allocated to every process. A process is responsible for computing the layer associated to every triangle of his subset. In order to do this, processes run the following algorithm.

```
Broadcast(p, root);
int pid = get_rank();  // process id.
int np  = get_size();  // # processes
int n   = p.n_edges(); // # edges
int sr;                // starting row
int ne;   // # edges allocated
calculate_chunk(pid, np, n, &sr, &ne);
bool M[ne][n];         // sub. matrix
int sc[ne];            // sub. counters
// Stage 1: Build M and sc
for (int i = 0; i < ne; i++) {
  sc[i] = 0;
  for (int j = 0; j < n; j++) {
    M[i][j] = p.edge(sr+i).sub_to
              (p.edge(j));
    sc[i] += M[i][j] ? 1 : 0;
  }
}

// Stage 2: Build layers
int ntp = n;  // # triangles processed
int l = 0;      // current layer
vector<int> cl; // current local layer
vector<int> cg; // current global layer
while (ntp > 0) {
  l++; cl.resize(0); cg.resize(0);
  p.lrep.add_layer(l);
  for (int i = 0; i < ne; i++) {
    if (sc[i] == 0) {
      cl.push_back(sr+i);
      sc[i] = -1;  // inserted
    }
  }
  AllGather(cl, cg);
  for (int i = 0; i < cg.size(); i++) {
    p.lrep.add_triangle(l, p.edge(cg[i]));
    for (int j = 0; j < ne; j++) {
      if (M[j][cg[i]]) {
        sc[j]--;
      }
    }
  }
  ntp -= cg.size();
}
```

At first, every process receives a copy of the polygon and calculates the subset of triangles allocated to it (from sr to $sr + ne - 1$). After this, the process computes the subordination matrix and the subordination counters of his subset of triangles. This computation is done without interacting with other processes. Next, the second stage starts. In this stage the layers of the polygon are calculated. Every process computes locally the triangles from his subset belonging to the current layer. Next, processes gather the current layer triangles found in all the processes by an *Allgather* collective operation. So that processes can decrement local subordination counters.

Before ending this section we want to do the following remarks:

- The *Broadcast*, *get_rank*, *get_size*, *Allgather* and *calculate_chunk* functions used in the algorithm are similar to functions defined in the MPI specification [Mpi94a].
- *Broadcast* and *Allgather* are synchronous collective operations. So, the parallel algorithm progresses at the pace of the slowest process — the others must wait for it. However, this is not a problem if the target hardware, as in our case, is a homogeneous cluster exclusively dedicated to the application execution. In fact, taking into account our target hardware, self-scheduling schemes as master-slave perform worse than our proposed solution.

Edges (number of processes)	Time	Speedup
1000 (1)	0,62	–
1000 (2)	0,51	1,22
1000 (4)	0,22	2,82
1000 (8)	0,18	3,44
5000 (1)	21,78	–
5000 (2)	9,12	2,39
5000 (4)	4,46	4,88
5000 (8)	2,43	8,96
10000 (1)	85,82	–
10000 (2)	42,32	2,03
10000 (4)	20,93	4,1
10000 (8)	9,91	8,66

Table 1. Execution results (in seconds).

- At algorithm termination the L-REP of the polygon is available to all the processes. This can be useful for future parallel operations on the polygon.

5. EXPERIMENTATION

In this section we describe the results from implementing the previous section algorithm using MPI. The target hardware consists of 8 AMD dual processors at 2.1 GHz connected by a 1 Gigabit/s Ethernet.

Table 1 shows the results of some sample executions. The left column contains the execution parameters, i.e., the number of edges of the polygon and the number of processes — or processors — used in the parallelization. If the number of processes is 1, the sequential algorithm is executed. The others columns show the execution time and the speedup obtained — calculated as the sequential execution time divided by the parallel execution time.

As can be seen the results are excellent. Even for the polygon with 1000 edges, which sequential execution only takes 0,62 seconds, good performance in terms of speedup has been obtained. For the polygons with 5000 and 10000 edges a super speedup is obtained — greater than n if n processors are used. This fact is normal when parallelizing algorithms with great memory needs, as in our case due to the subordination matrix, because parallel distributed hardware not only offers more CPU cycles, but also more cache and main memory.

Lastly to note some information not shown in Table 1:

- The time needed to broadcast the polygons is negligible compared to the total execution time.
- The speedup obtained in the two stages of the parallel algorithm is similar.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have described the parallelization of the algorithm used for constructing the L-REP of a polygon. As described in the previous section, the execution results are excellent, specially when representing big polygons. Big polygons can be frequently found, as in complex fonts or GIS, so the parallelization is quite useful. Furthermore, this algorithm is extensible to 3D where the size of the objects can be very big.

In the future we would like to apply parallel processing techniques to some applications of the L-REP such as Boolean operations or plain location.

7. ACKNOWLEDGEMENTS

This work has been partially granted by the Ministry of Science and Technology of Spain and the European Union by means of the ERDF funds, under the research projects TIC-2001-2099-C03-03 and TIN-2004-06326-C03-03.

8. REFERENCES

[Fei95a] Feito, F.R.; Torres J.C., “Orientation, simplicity and inclusion test for planar polygons”, Computer & Graphics 19, pp. 595–600, 1995.

[Fei97a] Feito, F.R.; Torres J.C., “Inclusion test for general polyhedra”, Computer & Graphics 21, pp. 64–77, 1997.

[Fei99a] Feito, F.R.; Rivero M. y Rueda, A.J., “Boolean representation for general planar polygons”, Proceedings of the WSCG’99, pp. 87–92, 1999.

[Mpi94a] Message Passing Interface Forum, “MPI: A Message-Passing interface standard”, International Journal of Supercomputer Applications, 8(3/4), pp. 165–414, 1994.

[Riv00a] Rivero, M.L.; Feito, F.R., “Boolean operations on general planar polygons”, Computer & Graphics 24, pp. 881–896, 2000.

[Rue02a] Rueda, A.J.; Feito, F.R. y Rivero, M., “A triangle-based representation for polygons and its applications”, Computer & Graphics 26, pp. 805–814, 2002.