ABSTRACT
This technical report is an accompaniment to the paper in [2]. Thus, all algorithms, theorems, propositions, ...etc referenced below should be matched with the corresponding ones in [2].

1. MEMORY PERFORMANCE ANALYSIS OF ALG. 1 RE-VISITED
The following is a discussion how the two assumptions below can be simulated for free in a strictly computational model such as the RAM model.

Assumption 1. Given an arbitrary point in the plane, it is assumed that the query whether or not this point belongs to $P$ can be answered in constant time.

In [1], it is shown that this is possible by pre-processing on $P$. For all vertices of the input polygon, let $y_{\text{max}}$ and $x_{\text{max}}$ denote the maximum vertex coordinates in $y$ and $x$ respectively, and $y_{\text{min}}$ and $x_{\text{min}}$ denote the minimum vertex coordinates in $y$ and $x$ respectively. Without loss of generality, we may assume that $y_{\text{min}} = x_{\text{min}} = 0$ (recall that the polygon is uniquely identified up to translation by the pivot $v_0$). For every $k = 0, \ldots, y_{\text{max}}$, say, we produce pairs of “boundary” points of the form $(x_k, k)$ and $(x'_k, k)$. For a given such $k$, an arbitrary point $(h, k)$ of the plane belongs to $P$ iff $x_k \leq h \leq x'_k$, so that the query is answered in $\Theta(1)$ time. Note that only boundary points will need to be stored, which requires a two dimensional integer array $E$ of size $2 \times y_{\text{max}}$, such that $E_{0,k} = x_k$ and $E_{1,k} = x'_k$, for all $k = 0, \ldots, y_{\text{max}}$. If we let $D = \max(y_{\text{max}}, x_{\text{max}})$, this process requires $O(mn^2D)$ operations in total, as part of pre-computation ([1]).

Assumption 2. Redundancies in vector computations arising in any one iteration of Alg. 1 are assumed to be eliminated before the start of the next iteration. It is also assumed that one can eliminate each redundant vector computation in constant time.

One may come across redundancies in vector computations as follows: within each iteration embedded in the $i$'th run of Step 2 – such as in 2.1 or 2.2 – it may be possible to generate two distinct points $u$ and $u'$, using two distinct positive integers $k, k' \leq n_i$, such that $u + ke_i = u' + k'e_i \in P$. The same can be argued for Step 3. (Notice for example, the vectors $\gamma_3 = \alpha_0$ and $\gamma_4 = \beta_2$ in Sect. 2.3. In Thm. 1 it is assumed that such redundancies in vector computations do not carry from one iteration to the next. This may be achieved in practice using a flag matrix, say, implemented as a two dimensional array $F$, of total size $x_{\text{max}} \cdot y_{\text{max}}$. All flags are initialised to “off”. One then flags by “on” the location in the matrix corresponding to the Cartesian coordinates of any one point, the first time this point is produced and stored. If further vector computations were to produce the same vector point, the flag prevents one from storing it again in the set $A_i$. By this, each redundant vector point is eliminated using $\Theta(1)$ time.

2. REFERENCES