Abstract

We investigate architectural schemes, generalizing that of existing graphics engines, supporting fast rendering of polygon meshes. A mesh defined on \( n \) vertices is rendered by sending vertices down a graphics pipeline, after which some are stored in a stack, to be removed when no longer needed. Only individual polygons whose vertices are present in the stack may be rendered. The storage cost of the mesh rendering is the size of the stack required to store mesh vertices. This may be significantly less than \( n \). The time cost of the mesh rendering is the number of vertices sent down the graphics pipeline. If a large enough stack is available, it suffices to send each vertex once. If only a small stack is available, some vertices may have to be sent more than once, so a time/space tradeoff exists.

With our architecture, a stack of size \( \Omega(\sqrt{n}) \) is sufficient to render any polygon mesh defined on \( n \) vertices, such that each vertex is sent only once through the graphics pipeline (time cost = \( n \)). We provide an algorithm which generates an appropriate “rendering sequence” of commands for any given mesh. Moreover, we show that no algorithm can do better, i.e. \( \Omega(\sqrt{n}) \) is a lower bound.

Some \( n \)-vertex meshes may be rendered using a stack whose size is significantly less than \( \Theta(\sqrt{n}) \). An algorithm generating a minimum-time rendering sequence requiring the minimum stack size is an open question. We provide an approximation: If it is possible to render a polygon mesh in minimum time with a stack of size \( S \), we provide an algorithm which generates a minimum-time rendering sequence requiring a stack of size \( \leq 2S \log_{3/2} n \).

If only a stack of size \( k \) is available, we provide an algorithm generating a rendering sequence requiring a stack of size \( \leq k \), such that at most \( n(1 + c/k) \) vertices must be sent through the pipeline, for some constant \( c \).

1 Introduction

In computer graphics and geometric modeling applications, it is common to approximate freeform smooth objects as polyhedra. The polyhedra may then be manipulated and rendered (drawn) efficiently using hardware-implemented routines, incorporating visible surface determination, shading, etc. The set of polygons defining a polyhedron is commonly known as a polygon mesh, having the topology of a planar graph.

Polygon meshes, and in particular, triangle meshes, are so common that leading graphics software libraries (e.g. GL [7], OpenGL [10], IGL [3]) provide function calls dedicated to mesh rendering, and some of these are supported in the hardware of some machines (e.g. SGI, Intel i860). For example, the SGI hardware dedicates three registers to vertex storage. A triangle mesh is rendered
using the GL graphics library by sending a sequence of vertices through the graphics pipeline, and a triangle is drawn automatically between every three consecutive vertices of the sequence, the three registers being used as a queue. Thus, a sequence of \( m \) vertices (perhaps with repetitions) specifies \( m - 2 \) triangles. Since at least one vertex must be supplied to render each triangle, ideally, only one vertex should be sent through the graphics pipeline for each triangle. As, by Euler’s theorem ([12], p. 19), the number of triangles in a mesh may reach up to twice the number of vertices, the GL scheme requires sending each vertex twice on the average, assuming the mesh can be specified in one sequence.

The time cost of the rendering operation is the number of vertices sent down the graphics pipeline, as these require expensive geometric projection and clipping operations. However, the class of triangle meshes that may be specified in just one vertex sequence (also called “rendering sequence”) without repetitions is extremely limited. These are “strip” like meshes (see Fig. 1(a)), also called \( \text{sequential} \) triangulations. To relax this constraint, the GL library allows the programmer to swap the contents of the two inner hardware registers at negligible cost (relative to sending a vertex through the pipeline). Interleaving \( \text{swap} \) commands among the vertices sent allows a larger class of \( \text{Hamiltonian} \) triangulations to be rendered at the cost of \( m + 2 \) vertices for \( m \) triangles. This is precisely the family of triangle meshes in which the triangles can be covered by a single path, along which each triangle appears only once, and only passages between triangles with common edges are allowed. These sequences are called a \( \text{Hamiltonian cover} \) of the mesh (see Fig. 1(b)). Unfortunately, not all triangle meshes are Hamiltonian, so an arbitrary mesh must be specified as a list of \( k \) vertex sequences, each beginning with the \( \text{begin} \) command and ending with the \( \text{end} \) command (see Fig 1(c)). The rendering time cost in this case is \( m + 2k \) vertices for \( m \) triangles. Minimizing \( k \) is NP-hard (by reduction to the Hamiltonian path problem), and approximating the minimum is an open question. Arkin et. al [2] show how to construct Hamiltonian triangulations on any planar point set, how to determine if a polygon has a Hamiltonian triangulation, and how to produce \( \text{sequential} \) covers of a given triangle mesh, within a constant factor of the optimum. Akeley et al [1] provide a greedy algorithm, based on heuristics, for the computation of a rendering sequence for a given triangle mesh. It is not clear how close to optimum the cost of the generated rendering sequence is.

In this paper we propose extending the existing hardware architectures by increasing the number of registers dedicated to the rendering process. This enables one to “remember” more of the vertices which have already travelled down the graphics pipeline, and allows the rendering of polygons with a large number of edges as an atomic operation. The former enables one to reduce the number of vertices sent in order to render a polygon mesh. In particular we show that \( 13.35 \sqrt{n} \) registers suffice to render any polygon mesh defined on \( n \) vertices in minimum time \( n \) (as opposed to Hamiltonian triangle meshes, which may require up to \( 2n \) vertices to be sent), and provide an algorithm which generates the appropriate rendering sequence for a given mesh. Moreover, this bound is tight up to a constant factor, in the sense that there exist polygon meshes on \( n \) vertices that a machine with less than \( 1.649 \sqrt{n} \) memory cannot render by sending each vertex only once.

Some polygon meshes possess a topology which might allow minimum cost rendering with a stack size considerably less than \( O(\sqrt{n}) \). We provide an algorithm which generates a rendering sequence, such that, if a stack of size \( S \) is sufficient for that mesh, the rendering sequence consumes at most \( 2S \log_{3/2} n \) storage.

If we are willing to trade off memory size in favor of rendering time, we show that any polygon mesh on \( n \) points can be rendered in \( n(1 + c/k) \) time, for some constant \( c \), if the memory size is \( k \), and provide an algorithm generating the rendering sequence. This is an approximation scheme enabling as close to minimum time rendering as desired, at the expense of extra storage.
Figure 1: Rendering with 3 registers: (a) Sequential triangle mesh. The equivalent GL rendering sequence is \texttt{begin,1,2,3,4,5,6,7,8,9,end}. (b) Hamiltonian triangulation. The equivalent GL rendering sequence is \texttt{begin,1,2,3,4,5,6,7,8,swap,9,swap,10,5,11,3,1,end}. (c) Non-Hamiltonian triangulation requires \( k = 2 \) vertex sequences to cover it. The equivalent GL rendering sequence is \texttt{begin,4,2,7,swap,5,swap,3,1,end,begin,9,7,8,swap,5,swap,3,6,end}.

2 Some Planar Graph Theory

3D polygon meshes are embedable in the plane, hence are planar graphs, in which every edge participates in a face (polygon) of the graph (mesh). When dealing with planar graphs, we can make use of the celebrated \textit{planar separator theorem} and its variants. We say that a class of graphs has a \((f(n), g(n))\)-\( \beta \)-separator for \( 1/2 \leq \beta < 1 \), if for any graph \( G(V, E) \) of \( n \) vertices \( V \) in the class, \( V \) can be partitioned into three sets \( U, S, W \) such that no edge in \( E \) joins a vertex in \( U \) with a vertex in \( W \), \((1 - \beta)f(n) \leq |U| \leq \beta f(n) \) and \(|S| < g(n)\). When \( f(n) = n \), we say simply that \( S \) is a \( \beta \)-\textit{separator} of \( G \).

Many problems concerning graphs in the class can be solved efficiently using the divide-and-conquer method if the vertex set \( S \) is small enough and can be computed efficiently. We use the following separator theorems:

\begin{theorem} ([9, 5]) The class of planar graphs has a \((n, \sqrt{6n})\) 2/3-separator which can be computed in \( O(n) \) time. \end{theorem}

The original theorem of Lipton and Tarjan [9] obtained \( g(n) = \sqrt{8n} \), which was later improved by Djidjev [5] to \( g(n) = \sqrt{6n} \). These functions are the best possible up to a constant factor for separators of planar graphs, as the following theorem asserts:

\begin{theorem} There exist classes of planar graphs which do not have a \((n, 1.649\sqrt{n}) (1/2 + o(1))\)-separator. \end{theorem}

Theorem 2 is a simple extension of the lower bound of \( 1.55\sqrt{n} \) on the 2/3-separator size proved by Djidjev [5], obtained on a triangulation of the sphere. The first lower bound of \( \sqrt{n} \) on the 2/3-separator size was obtained in the original paper by Lipton and Tarjan [9].

Theorem 1 implies the existence of a \( O(n) \) procedure \texttt{separate}(G) which, given a planar graph \( G \) on \( n \) vertices, computes a triple \(< U, S, W > \) such that \( S \) 2/3-separates \( G \) to \( U \) and \( W \) and \(|S| \leq \sqrt{6n} \).

\footnote{The exact constant is \( \sqrt{\frac{3}{2}} \)}. 

Technion - Computer Science Department - Technical Report CIS9515 - 1995
push(3) push(5) push(7) push(2) push(1) draw(1,2,3) draw(2,3,5) draw(2,5,7) pop push(4) draw(2,4,7) pop pop push(8) push(6) draw(3,5,8) draw(3,6,8) draw(5,8,7) pop push(9) draw(7,8,9) pop pop pop pop pop

Figure 2: A minimum-time rendering sequence for a polygon mesh on 9 vertices with memory size \( k = 5 \). Each vertex is pushed only once.

3 Minimum Time Polygon Mesh Rendering

We propose an architecture for rendering polygon meshes, where the machine is equipped with registers for storing mesh vertices in a stack mechanism. Three operations are possible: push(\( v \)) - send a vertex \( v \) down the graphics pipeline into the vertex stack, draw(\( i_1, \ldots, i_k \)) - draw a polygon incident on the vertices in entries \( i_1, \ldots, i_k \) of the vertex stack, pop - remove the top vertex from the stack. push(\( V \)) for a vertex set \( V \) is shorthand for the sequence (push(\( v \) : \( v \in V \)), and pop(\( k \)) shorthand for a sequence of \( k \) pop operations. The dominant cost is incurred by the push operation.

Just as the swap operation of the existing GL scheme is cheap (it involves sending only one bit down the pipeline), we make a similar assumption for the pop operation. The draw operation is slightly more expensive, as it would involve transmitting a few integers, but no computation.

A rendering sequence for a mesh \( M \) is a sequence of push, draw or pop operations, such that after performing these operations, all polygons of \( M \) have been drawn. The cost of push(\( V \)) is \( |V| \). The time cost of a rendering sequence is the total cost of the push operations in the sequence. See Fig. 2 for an example. The minimum time cost of rendering a polygon mesh defined on \( n \) vertices is \( n \). We now show that with a vertex stack of size \( 13.35 \sqrt{n} \), it is possible to render any polygon mesh in minimum time. The minimum time cost rendering sequence is generated by the following recursive procedure taking advantage of the planar separator theorem (Theorem 1).
minimum_time_render(M(V))
/* Generate minimum-time rendering sequence for mesh M on vertex set V */
{
   /* Global stack in initialized to be empty. */
   if (|V|==0)
      for all polygons p of M on vertices in Stack not yet drawn
         output "draw(" index(v1(p)) index(v2(p)) .. index(vk(p)) ")"
   else {
      <U,S,W> := separate(M);
      output "push(" S ")";
      minimum_time_render(M(U)); /* M(U) is the subgraph of M induced by U */
      minimum_time_render(M(W));
      output "pop(" |S| ")";
   }
}

Theorem 3 Algorithm minimum_time_render generates a rendering sequence for M(V) requiring a stack of size \( \leq 13.35 \sqrt{n} \). No vertex of V is pushed more than once. The algorithm runs in \( O(n \log n) \) time and \( O(|V|) \) space.

Proof: The height of the stack used by the rendering sequence generated by the algorithm is the sum of the sizes of the separators computed along the worst path of the recursion tree. By Theorem 1, it is bounded by

\[
\text{Height} \leq \sqrt{6n} + \sqrt{6(2n/3)} + \sqrt{6(4n/9)} + \cdots
\]
\[
< \sqrt{6n} \sum_{i=1}^{\infty} \frac{(2/3)^{i/2}}{2/3}
\]
\[
= \frac{\sqrt{6}}{1 - \sqrt{2/3}} \sqrt{n}
\]
\[
< 13.35 \sqrt{n}
\]

At each recursive stage, the algorithm partitions \( V \) into three disjoint sets \( U, S, W \). The vertices in \( S \) are pushed on the stack, and then the procedure applied on \( U \) and \( W \), implying that no vertex of \( V \) is pushed more than once.

The computation time consumed by minimum_time_render is dominated by the calls to separate. Since separate runs in \( O(|V|) \) time, the following inequality holds for the complexity \( C(n) \) of minimum_time_render:

\[
C(n) = O(n) + C(\beta_n n) + C((1 - \beta_n)n)
\]

where \( 1/3 \leq \beta_n \leq 2/3 \) for all \( n \). This implies \( C(n) = O(n \log n) \). The space consumed by minimum_time_render is dominated by the need to keep global track of the which polygons have, and have not, been drawn. This can be done with a global hash table of size proportional to the number of polygons in \( M \), namely \( O(|V|) \).

We now prove that this upper bound on the memory required for minimum-time rendering is tight up to a constant factor.

Theorem 4 Any machine with the capability of rendering all polygon meshes on \( n \) vertices in minimum time requires a stack of size at least \( 1.649 \sqrt{n} \).
Proof: Assume a machine capable of rendering any polygon mesh in minimum time. Given any polygon mesh $M$, the state of the rendering machine at any stage during rendering may be described by the three disjoint vertex sets: $U =$ vertices already removed from memory, $S =$ vertices in memory, $W =$ vertices not yet inserted into memory. Since the vertices of $U$ will never be inserted into memory again (otherwise the rendering will not have minimum time), all edges incident on vertices of $U$ must have been rendered already. On the other hand, since the vertices of $W$ have not yet been seen by the machine, no edge incident on vertices of $W$ has been rendered yet. This means that $G$ does not contain any edge between a vertex of $U$ and a vertex of $W$, implying that $S$ separates $G$.

Stopping the rendering process after the first push operation where $|W| < n/2$ implies that certainly $|U| < n/2$ before that operation ($|W|$ was previously $\geq n/2$). This still holds after the push operation, as these vertices are then in $S$, therefore $S$ is a 1/2-separator of $G$. By Theorem 2, there exist polygon meshes such that the size of $S$ must be $\geq 1.649\sqrt{n}$.

Note: The lower bound of Theorem 2 holds for triangle meshes, hence Theorem 4 is true even for machines capable of rendering only triangle meshes.

4. Minimal Storage Polygon Mesh Rendering

In the previous section, we proved that any planar graph with $n$ vertices may be rendered in minimum time $n$ using a stack of size $O(\sqrt{n})$. This was due largely to the fact that a well-balanced separator of size $O(\sqrt{n})$ is guaranteed to exist for any planar graph with $n$ vertices, and may be computed efficiently.

However, for many planar graphs, much smaller separators may exist, implying that a significantly smaller stack might suffice for minimum time rendering. Hence, it is advantageous to apply algorithms for computing minimum separators. A minimum $\beta$-separator of a graph $G$ is a $\beta$-separator of $G$ of minimum size. Denote by $Sep_{\min}(G(V), \beta)$ the size of the minimum $\beta$-separator of $G(V)$. Since separator computations will typically be done offline, in a preprocessing step, the complexity of the separator-computing algorithm is not critical, as long as it is polynomial. Computing the minimum $\beta$-separator of a $n$-vertex planar graph for any constant $1/2 \leq \beta \leq 1$ is believed to be NP-hard [11], so an efficient solution currently is only approximate, i.e. given a planar graph $G(V)$ and $1/2 \leq \beta \leq 1$, a separator of size $\leq K(n)Sep_{\min}(G(V), \beta)$ is computed, where $K(n)$ is an approximation factor. In a series of works, approximations to the optimum separator problem have been proposed for $K(n) = O(\log n)$ [13], $K(n) = O(1)$ [4], and $K(n) = 2$ [6]. Call the algorithm of [6] approximate optimum separator. It runs in polynomial time.

Denote by $Stack_{\min}(G(V))$ the size of the minimum stack size required to render $G(V)$ in minimum time. We are not able to present an algorithm which produces a minimum-time rendering sequence for a planar graph using a stack of size $Stack_{\min}(G(V))$, but we are able to provide an approximation:

**Theorem 5** Applying algorithm $minimum\_time\_render$ to a $n$-vertex mesh $M$ with calls to $approximate\_optimum$ instead of $separate$ generates a rendering sequence requiring a stack of size $\leq 2Stack_{\min}(M)\log_{1/2} n$.

First, the following Lemma establishes a relation between the quantities $Sep_{\min}$ and $Stack_{\min}$.

**Lemma 1** Let $G(V)$ be a planar graph. For any subgraph $G(V')$ of $G$ induced by $V' \subset V$, and for any $1/2 \leq \beta \leq 1$, $Sep_{\min}(G(V'), \beta) \leq Stack_{\min}(G(V))$. 
Proof of Lemma: As already observed in the proof of Theorem 4, the contents of the stack of the rendering machine at any time during the rendering process defines a separator of $G(V)$, and, therefore, also of $G(V')$.

Consider the minimum-space rendering sequence of $G(V)$ (which requires a stack of size $Stack_{\text{min}}(G(V))$). For any $1/2 \leq \beta \leq 1$, it is possible to find a point during the rendering, in which $S'$, the subset of $V'$ in the stack, $\beta$-separates $G(V')$. By definition, the size of the entire stack at that moment is $\leq Stack_{\text{min}}(G(V'))$. $S'$ is a subset of the vertices in the stack, hence $|S'| \leq Stack_{\text{min}}(G(V'))$. The fact that $S'$ is a $\beta$-separator of $G(V')$ implies that $Sep_{\text{min}}(G(V'), \beta) \leq |S'|$, hence $Sep_{\text{min}}(G(V'), \beta) \leq Stack_{\text{min}}(G(V'))$.

Proof of Theorem: The proof of this theorem proceeds by an argument similar to that applied by Leighton and Rao [8] for their approximation scheme to the minimum cut linear arrangement problem on general graphs.

The stack size required by the generated rendering sequence is the sum of the separator sizes computed along the worst path of the recursion tree of algorithm minimum_time_render. Calling approximate_optimum_separator guarantees that the size of each such separator is $\leq 2Sep_{\text{min}}(G(V_i), \beta_i)$, for the appropriate $V_i \subset V$ and $\beta_i$, therefore, by Lemma 1, not more than $2Stack_{\text{min}}(G(V'))$. The depth of the recursion tree is $\leq \log_{3/2} n$, so the total stack size is $\leq 2Stack_{\text{min}} \log_{3/2} n$.

We note that $Sep_{\text{min}}(G(V), \beta)$ for many polygon meshes on $n$ vertices is close to the upper bound $O(\sqrt{n})$, so computing a minimum $\beta$-separator might not be a major improvement on an arbitrary $\beta$-separator.

5 Time/Space Tradeoffs

It is possible to use the same basic technique we used to generate a minimum-time rendering sequence for the entire mesh, but for much smaller pieces. This facilitates a much smaller vertex stack size, but increases the rendering time cost, as some vertices must be pushed more than once. To establish this, we need a finer version of the basic separator Theorem 1, dealing with separations to less balanced parts:

**Theorem 6** For any $g(n) = O(n)$, the class of planar graphs on $n$ vertices has a $(g(n), 12.5 \sqrt{g(n)})$ 2/3-separator which can be computed in $O(n)$ time.

**Proof:** Denote $\alpha = 12.5$. Given a planar graph $G(V)$, apply Theorem 1 recursively. At each recursion level, a vertex set is separated into two sets, of which one is chosen for further separation. This continues until a set of size less than $2g(n)/3$, but larger than $g(n)/3$, is reached. We prove inductively that the theorem holds at each recursion level.

The induction basis, for which $g(n) = n$, is implied by Theorem 1. Now assume that the induction hypothesis holds for a $m$-vertex planar graph $G(V)$, i.e. the size of the separator $S$ which yielded this set is $< \alpha \sqrt{m}$. Now separate $G(V)$, containing $m$ vertices, with a separator $S'$ of size $\sqrt{6m}$, to two vertex sets $V_1$ and $V_2$ of size $m_1$ and $m_2$, respectively. Assume $|S \cap V_1| = \alpha_1 \sqrt{m}$ and $|S \cap V_2| = (\alpha - \alpha_1) \sqrt{m}$.

Now $S_1 = S' \cup (S \cap V_1)$ is a separator of $G(V_1)$, and $S_2 = S' \cup (S \cap V_2)$ is a separator of $G(V_2)$. If, contrapositively, both $|S_1| > \alpha \sqrt{m_1}$ and $|S_2| > \alpha \sqrt{m_2}$, then $(\alpha_1 + \sqrt{6}) \sqrt{m} > |S_1| > \alpha \sqrt{m_1}$, and similarly $(\alpha - \alpha_1 + \sqrt{6}) \sqrt{m} > \alpha \sqrt{m_2}$, so $(\alpha + 2 \sqrt{6}) \sqrt{m} > \alpha \sqrt{m_1 + m_2}$. Now since $m_1 + m_2 = m$ and $m/3 \leq m_1, m_2 \leq 2m/3$, this implies $\sqrt{m_1} + \sqrt{m_2} > \frac{1 + \sqrt{6}}{\sqrt{3}} \sqrt{m}$, in turn implying $\alpha + 2 \sqrt{6} > \frac{1 + \sqrt{6}}{\sqrt{3}} \alpha$, contradicting the fact that $\alpha = 12.5$. 


Therefore, without loss of generality, we may assume that $|S_i| < a\sqrt{n_i}$ and proceed recursively to separate $V_i$, until a component of the required size is obtained.

$C(n)$, the complexity of this procedure, satisfies

$$C(n) \leq O(n) + C(2n/3)$$

implying $C(n) = O(n)$.  

Theorem 6 implies the existence of a $O(n)$ procedure $\text{separate}(G,k)$, which, given a planar graph $G(V)$ on $n$ vertices, computes a triple $<U,S,W>$ such that $S$ separates $G$ to $U$ and $W$, $(W = V - U - S)$, $k/3 \leq |U| \leq 2k/3$ and $|S| < 12.5\sqrt{k}$.

Given a polygon mesh $M$, using Theorem 6, we separate a small submesh $M'$ from $M$, use $\text{minimum_time_render}$ to generate a rendering sequence for the polygons defined on $M'$ (including the separator), and then discard $M'$. The process is continued with the remainder of the mesh (again including the separator).

Assume the vertex stack size is $k$. The following algorithm generates a rendering sequence for a polygon mesh under this constraint. $d$ is sufficiently large constant, whose exact value may be determined later.

```plaintext
render(M(V)) /* generate rendering sequence for mesh M on vertex set V */
{
    while (|V|!=0) {
        <U,S,W> := separate(M(V),d*k*k)
        minimum_time_render(M(U+S)); /* U+S is the union of U and S */
        V := W+S;
    }
}
```

**Theorem 7** Algorithm $\text{render}$ generates a rendering sequence requiring a stack of size $\leq k$, which pushes $\leq n(1 + c/k)$ vertices, for some constant $c$. The algorithm runs in $O(n^2/k^2)$ time.

**Proof:** At each iteration of the loop, $\text{minimum_time_render}$ pushes the vertices of $S$ once in order to render $M(U \cup S)$, but these are pushed again in the rendering of $M(W \cup S)$. The number of iterations is $O(n/k^2)$. By Theorem 6, at each iteration $|S| = O(k)$, so the number of vertices pushed at that iteration is $|U \cup S| \leq |U| + |S| = |U| + O(k)$. The total number of vertices pushed during $\text{render}$ is therefore $n + O(n/k^2)O(k) \leq n(1 + c/k)$, for some constant $c$. $\text{render}$ does not require a stack of size more than $k$ since this holds for $\text{minimum_time_render}$. Each iteration runs in $O(n)$ time, so the total run time is $O(n^2/k^2)$.

6 Conclusion

We have explored the advantages of extending the architecture of contemporary graphics engines to larger vertex stores, in order to render polygon meshes more rapidly. We have shown that any $n$-vertex mesh may be rendered in minimum time with storage cost $\Theta(\sqrt{n})$. We have also optimized this, and shown how to gracefully trade off memory for time.

Our rendering model assumes that the dominant time cost of rendering a polygon is incurred at the geometric stage of the rendering pipeline. For some machines it is not obvious that this is the case. The bottleneck of the rendering pipeline might be elsewhere, e.g. in the polygon scan conversion. In this case the projected area of the polygons is important, and larger polygons are to be considered more complex than smaller ones.
Acknowledgements

We thank Noga Alon, Estie Arkin, Guy Even, Joe Mitchell, Seffi Naor and Satish Rao for helpful discussions and comments on the topic of this paper.

References


