3D Rectangulations and Geometric Matrix Multiplication

Peter Floderus[1], Jesper Jansson[2], Christos Levcopoulos[3], Andrzej Lingas[3](✉), and Dzmitry Sledneu[1]

[1] Centre for Mathematical Sciences, Lund University, 22100 Lund, Sweden
{pflo,Dzmitry}@maths.lth.se
[2] Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan
jj@kuicr.kyoto-u.ac.jp
[3] Department of Computer Science, Lund University, 22100 Lund, Sweden
{Christos.Levcopoulos,Andrzej.Lingas}@cs.lth.se

Abstract. The problem of partitioning an input rectilinear polyhedron P into a minimum number of 3D rectangles is known to be NP-hard. We first develop a 4-approximation algorithm for the special case in which P is a 3D histogram. It runs in $O(m \log m)$ time, where m is the number of corners in P. We then apply it to compute the arithmetic matrix product of two $n \times n$ matrices A and B with nonnegative integer entries, yielding a method for computing $A \times B$ in $\tilde{O}(n^2 + \min\{r_A r_B, n \min\{r_A,\ r_B\}\})$ time, where $\tilde{O}$ suppresses polylogarithmic (in n) factors and where r_A and r_B denote the minimum number of 3D rectangles into which the 3D histograms induced by A and B can be partitioned, respectively.

Keywords: Geometric decompositions · Minimum number rectangulation · Polyhedron · Matrix multiplication · Time complexity

1 Introduction

This paper considers two intriguing and at a first glance unrelated problems.

The first problem lies at the heart of three-dimensional computational geometry. It belongs to the class of *polyhedron decomposition* problems, whose applications range from data compression and database systems to pattern recognition, image processing, and computer graphics [7,13]. The problem is to partition a given rectilinear polyhedron into a minimum number of 3D rectangles. Dielissen and Kaldewai have shown this problem to be NP-hard [4]. In contrast, the problem of partitioning a rectilinear (planar) polygonal region into a minimum number of 2D rectangles admits a polynomial-time solution [7,10]. Formally, the NP-hardness proof by [4] is for polyhedra with holes, but the authors remark that the proof should also work for simple polyhedra. To the best of our knowledge,

Jesper Jansson: Funded by The Hakubi Project at Kyoto University.
Christos Levcopoulos: Research supported in part by Swedish Research Council grant 621-2011-6179.

H.-K. Ahn and C.-S. Shin (Eds.): ISAAC 2014, LNCS 8889, pp. 65–78, 2014.
DOI: 10.1007/978-3-319-13075-0_6

no non-trivial approximation factors for minimum rectangular partition of simple rectilinear polyhedra are known, even in restricted non-trivial cases such as that of a 3D histogram (a natural generalization of a planar histogram, see Section 2).

The second problem we consider is that of multiplying two $n \times n$ matrices. There exist fast algorithms that do so in substantially subcubic time, e.g., a recent one due to Le Gall runs in $O(n^{2.3728639})$ time [8], but they suffer from very large overheads. On the positive side, input matrices in real world applications often belong to quite restricted matrix classes, so a natural approach is to design faster algorithms for such special cases. Indeed, efficient algorithms for *sparse* matrix multiplication have been known for long time. In the Boolean case, despite considerable efforts by the algorithms community, the fastest known combinatorial algorithms for Boolean $n \times n$ matrix multiplication barely run in subcubic time (in $O(n^3(\log\log n)^2/(\log n)^{9/4)})$ time [1], to be precise), but much faster algorithms for Boolean matrix product for restricted classes of Boolean matrices have been developed [3,5,9]. For example, when at least one of the input Boolean matrices admits an exact covering of its ones by a relatively small number of rectangular submatrices, the Boolean matrix product can be computed efficiently [9]; similarly, if the rows of the first input Boolean matrix or the columns of the second input Boolean matrix can be represented by a relatively cheap minimum cost spanning tree in the Hamming metric (or its generalization to include blocks of zeros or ones) then the Boolean matrix product can be computed efficiently by a randomized combinatorial algorithm [3,5].

Our first contribution is an $O(m \log m)$-time, 4-approximation algorithm for computing a minimum 3D rectangular partition of an input 3D histogram with m corners. It works by projecting the input histogram onto the base plane, partitioning the resulting planar straight-line graph into a number of 2D rectangles not exceeding its number of vertices, and transforming the resulting 2D rectangles into 3D rectangles of appropriate height. Importantly, the known algorithms for minimum partition of a rectilinear polygon with holes into 2D rectangles [7,10] do not yield the aforementioned upper bound on the number of rectangles in the more general case of planar straight-line graphs.

Our second contribution is a new technique for multiplying two matrices with nonnegative integer entries. We interpret the matrices as 3D histograms and decompose them into blocks that can be efficiently manipulated in a pairwise manner using the interval tree data structure. Let A and B be two $n \times n$ matrices with nonnegative integer entries, and let r_A and r_B denote the minimum number of 3D rectangles into which the 3D histograms induced by A and B can be partitioned. By applying our 4-approximation algorithm above, we can compute $A \times B$ in $\tilde{O}(n^2 + r_A r_B)$ time, where $\tilde{O}$ suppresses polylogarithmic (in n) factors. Furthermore, by using another idea of slicing the histogram of A (or B) into parts corresponding to rows of A (or columns of B) and measuring the cost of transforming a slice into a consecutive one, we obtain an upper bound of $\tilde{O}(n^2 + n \min\{r_A,\ r_B\})$. We also give a generalization of the latter upper bound in terms of the minimum cost of a spanning tree of the slices, where the distance between a pair of slices corresponds to the cost of transforming one slice into the other.

Organization: Section 2 presents our 4-approximation algorithm for a partition of a 3D histogram into a minimum number of 3D rectangles. Section 3 presents our algorithms for the arithmetic matrix product. Section 4 concludes with some final remarks.

2 3D Histograms and Their Rectangular Partitions

A 2D histogram is a polygon with an edge e, which we call the *base* of the histogram, having the following property: for every point p in the interior of histogram, there is a (unique) line segment perpendicular to e, connecting p to e and lying totally in the interior of the histogram. In this paper, we consider orthogonal histograms only. For simplicity, we consider the base of a histogram as being horizontal, and all other edges of the histogram lying above the base. In this way, a 2D histogram can also be thought of as the union of rectangles standing on the base of the histogram.

A *3D histogram* is a natural generalization of a 2D histogram. To define a 3D histogram, we need the concept of the "base plane", which for simplicity we define as the horizontal plane containing two of the axes in the Euclidean space. A 3D histogram can then be thought of as the union of rectilinear 3D rectangles, standing on the base plane. The *base of the histogram* is the union of the lower faces (also called *bases*) of all these rectangles.

Definition 1. *A 3D histogram is a union of a finite set C of rectilinear 3D rectangles such that: (i) each element in C has a face on the horizontal base plane; and (ii) all elements in C are located above the base plane.*

(In the literature, what we call a 3D histogram is sometimes termed a 2D histogram or a 1D histogram when used to summarize 2D or 1D data, respectively [12].)

By a *rectangular partition* of 3D histogram P, we mean a rectilinear partition of P into 3D rectangles. In Section 2.2 below, we consider the problem of finding a rectangular partition of a given 3D histogram P into as few 3D rectangles as possible. We present a 4-approximation algorithm for this problem with time complexity $O(m \log m)$, where m denotes the number of vertices in P. The algorithm partitions P into less than m' 3D rectangles, where m' is the number of vertices in the vertical projection of P (i.e., $m' < m$), by applying a subroutine described in Section 2.1 that partitions any rectilinear planar straight-line graph (PSLG) with m' vertices into less than m' 2D rectangles. Finally, the approximation factor is derived by observing that any rectangular partition of P must contain at least $m'/4$ 3D rectangles.

2.1 Partitioning a Rectilinear PSLG into 2D Rectangles

The problem of partitioning a rectilinear polygon into rectangles in two dimensions has been well studied in the literature [7,10]. An optimal solution for this problem can be computed in polynomial time [7,10]. However, to use the result in 3D, we need a bound on the number of produced rectangles, expressed in

terms of the number of vertices. Therefore, it is not so crucial for our purposes to compute an optimal solution for the 2-dimensional problem, but instead, we need to partition planar straight-line graphs (PSLGs) into at most m' rectangles, where m' denotes the number of vertices in the input PSLG. We will show that a simple algorithm suffices to obtain this bound.

Since this subsection considers 2D only, we use the term "horizontal" for line segments parallel to the X-axis. By "vertical" lines, we mean lines or line segments parallel to the Y-axis. Each vertex in the planar graphs in our application has degree 2, 3, or 4.

Definition 2. *A planar straight-line graph (PSLG) $PG = (V, E)$, as used in this paper, is a planar graph where every vertex has an x- and a y-coordinate. Each edge is drawn as a straight line segment, all edges meet at right angles, and each vertex has degree 2, 3, or 4. A rectangular partition of PG is a partition $R = (V \cup V_R, E \cup E_R)$ that adds edges and vertices to PG so that R is still a PSLG while every face in R is a rectangle.*

Given a PSLG PG, we denote $m' = |V|$. We say that a vertex v of PG is *concave* if it has degree 2, its two adjacent edges are perpendicular to each other, and the corner at v which is of 270 degrees does not lie in the outer, infinite face of PG. Any vertex which is not concave is called *convex*.

We use a sweep line approach to generate a partition into less than m' rectangles. We perform a horizontal sweep with a vertical sweep line [2], using the vertices of PG as event points. Whenever the sweep line reaches a concave vertex v, we insert into the graph PG a vertical line segment s connecting v to the closest edge of PSLG upwards or downwards, thus canceling the concavity at v and transforming v into a convex vertex of degree 3. Hence, if there was already an edge of PG below v, then the new segment s is inserted above v, otherwise it is inserted below v. To preserve the property that the resulting graph is still a PSLG, the other endpoint of s may have to become a new vertex of the PSLG. This is a standard procedure for trapezoidation; see, e.g., [2] for more details. After the sweep is complete, all concave vertices have been eliminated. (Remark: In a special case it may happen that two concave vertices with the same x-coordinate are connected by a single vertical segment that is disjoint from the rest of the input PSLG. In this case, the plane sweep algorithm will produce this segment. Thus, no two segments produced by the algorithm overlap or touch each other.)

The correctness of the algorithm is easy to see: it eliminates all concave corners of PG by adding vertical line segments. Hence, in the resulting PSLG, each face, except for the outer face, is a rectangle. The running time of this algorithm is dominated by the cost of the plane sweep, which is $O(m' \log m')$ according to well-known methods in computational geometry; see, e.g., [2].

We need to relate the number of vertices in the input PSLG to the number of 2D rectangles. This is done in the following lemma:

Lemma 1. *Any PSLG $PG = (V, E)$ with $|V| = m'$ and minimum vertex degree 2 can be partitioned into b rectangles with $b < m'$ using $O(m' \log m')$ time.*

Proof. Let R denote the set of rectangles in the rectangular partition produced by the plane sweep algorithm described above. We use a "charging scheme" to prove the stated inequality. The charging scheme starts by giving each vertex $v \in V$ four tokens; thus, a total of $4m'$ tokens are used. Each vertex v then distributes its tokens in a certain way to the rectangles in R that are adjacent to v. We will show that every rectangle in R receives at least four tokens. Since we started by giving a total of $4m'$ tokens to the vertices, this will prove that there exist at most m' rectangles, and thus $b \leq m'$. Moreover, vertices adjacent to the outer face do not give away more than three tokens. We will thus obtain the strict inequality $b < m'$.

Now, we describe the details of the charging scheme. (More explanations and illustrating figures are included in the full version.) Let v be any vertex of V. The vertex v gives one token to each rectangle r in R which in any way is adjacent to it, with one exception. The exception occurs when v is a concave vertex; then, v is partitioned by a vertical segment e_r added by the algorithm. This segment partitions the three quadrants at the concave corner around the vertex so that one rectangle occupies one quadrant and one occupies the two others. Then v distributes two tokens to the new rectangle occupying only one quadrant, which therefore has a corner at v, and only one token to each one of the other rectangles of R adjacent to v.

We now show that each rectangle receives at least four tokens. Let r be any rectangle in R. First note that each vertical segment added by the algorithm has at least one endpoint at a vertex in V. Moreover, for any rectangle r in R, each of the vertical sides of r includes at least one vertex of V. Therefore, each rectangle is adjacent to at least two vertices of V. We distinguish three cases, depending on the number of vertices of V adjacent to r. Observe that the adjacencies are not necessarily at the corners of r.

- Case 1: r is adjacent to at least four vertices of V. Since r will receive at least one token from each of them we are done.
- Case 2: r is adjacent to precisely three vertices of V. Then at one of the vertical sides of r there is only one vertex of V. Moreover, this vertex v must be at a corner of r and fulfills the criteria for giving two tokens to r. The remaining two adjacent vertices of V give at least one token each, so we are done.
- Case 3: r is adjacent to precisely two vertices of V. This must mean that both vertical sides of r are segments added by the algorithm, and that one of the endpoints of each of these sides is a vertex of V at a corner of r. This corresponds to the condition for receiving two tokens mentioned earlier. So in total, r receives four tokens from the two corners, and we are done. □

2.2 Partitioning a 3D Histogram into 3D Rectangles

We now explain how to obtain the projected PSLG from the 3D histogram P and how to use the rectangular partition of this PSLG to yield a good partition into 3D rectangles.

Definition 3. *The planar projection PP is an orthogonal projection of the input 3D histogram P along the "down" direction onto the base plane in Definition 1.*

We can interpret PP as a PSLG where each corner and each subdividing point on a line segment corresponds to a vertex. The edges naturally correlate to the connecting line segments between vertices. Each vertex in PP is the vertical projection of at least two vertices of P. Two edges of the 3D histogram may partially overlap in the 2D projection, but the edges in the 2D projection are considered as non-overlapping. Thus, an edge of the 3D histogram may split into several edges in the 2D projection, since vertices should only appear as endpoints of edges.

Remark 1. Every vertex in PP must have at least two neighbors. This follows from the fact that each vertex of P (and of any orthogonal polyhedron) has at least two incident horizontal edges. It may happen that some vertex of PP is the vertical projection of up to four vertices of P, so those four vertices of P may have a total of eight neighbors in P. But since PP is an orthogonal PSLG, no vertex of PP has more than four neighbors.

Now we are ready to show the main theorem of this section.

Theorem 1. *For any 3D histogram P with m corners, a 4-approximation R of a partition of P into as few 3D rectangles as possible can be computed in $O(m \log m)$ time.*

Proof. We use the projection in Definition 3, let $PG = PP$, and apply Lemma 1 to compute a planar partition R'. The final 3D partition R is obtained from R' by reversing the projection so that each 2D rectangle corresponds to the top of a 3D rectangle in R.

To analyze the approximation factor, denote the number of 3D rectangles in an optimal solution R^* by OPT and the number of 3D rectangles produced by the algorithm described above by b. We denote by m' the number of vertices in PP. By Lemma 1, we have $b < m'$ since each 2D rectangle corresponds to one 3D rectangle. Every vertex of P must be adjacent to at least one vertical edge of a 3D rectangle in R^*. Hence, each vertex in PP has to be at a corner of the vertical projection of at least one 3D rectangle in R^* onto the base plane. Since each 3D rectangle in R^* only has 4 vertical edges, its vertical projection can be adjacent to at most 4 vertices of PP. It follows that $m' \leq 4OPT$ and $b < m' \leq 4OPT$.

Since the projection can be obtained by contracting each corner in P and all of its vertical neighbors into one vertex, the projection can be implemented in $O(m)$ time. Thus, the $O(m \log m)$-term from Lemma 1 will dominate the time complexity. □

3 Geometric Algorithms for Arithmetic Matrix Product

3.1 Geometric Data Structures and Notation

Our algorithms for arithmetic matrix multiplication use some data structures for interval and rectangle intersection. An *interval tree* is a leaf-oriented binary

search tree that supports intersection queries for a set Q of closed intervals on the real line as follows:

Fact 1 [11]. *Suppose that the left endpoints of the intervals in a set Q belong to a subset $\mathcal{U}$ of real numbers of size l and $|Q| = q$. An interval tree T of depth $O(\log l)$ for Q can be constructed in $O(l + q \log lq)$ time using $O(l + q)$ space. The insertion or deletion of an interval with left endpoint in $\mathcal{U}$ into T takes $O(\log l + \log q)$ time. The intersection query is supported by T in $O(\log l + r)$ time, where r is the number of reported intervals.*

Remark 2. The interval tree of Fact 1 ([11]) can easily be generalized to the weighted case, where with an interval to insert or delete an integer weight is associated. It can be done by maintaining in each node of the interval tree the sum of weights of intervals whose fragments it represents. In effect, the generalized interval insertions or deletions as well the intersection query have the same time complexity as those in Fact 1. Moreover, the generalized interval tree supports a weight intersection query asking for the total weight of the intervals containing the query point in $O(\log l + \log q)$ time.

We use the following data structure, easily obtained by computing all prefix sums:

Fact 2. *For a sequence of integers $a_1, a_2, \ldots, a_n$, one can construct a data structure that supports a query asking for reporting the sum $\sum_{k=i}^{j} a_k$ for $1 \leq i \leq j \leq n$ in $O(1)$ time. The construction takes $O(n)$ time.*

In the rest of the paper, A and B denote two $n \times n$ matrices with nonnegative integer entries, and C stands for their matrix product. We also need the following concepts:

1. For an $n \times n$ matrix D with nonnegative integer entries, consider the $[0, n] \times [0, n]$ integer grid whose unit cells are in one-to-one correspondence with the entries of$D$. The grid cell between the horizontal lines $i - 1$ and i (counting from the top) and vertical lines $j - 1$ and j (counting from the left) corresponds to $D_{i,j}$ (see Fig. 1a). Then, $his(D)$ stands for the 3D histogram whose base consists of all unit cells of the $[0, n] \times [0, n]$ integer grid corresponding to positive entries of D and whose height over the cell corresponding to $D_{i,j}$ is the value of $D_{i,j}$ (see Fig. 1b).
2. For the $n \times n$ matrix D, nonnegative integers $1 \leq i_1 \leq i_2 \leq n$, $1 \leq k_1 \leq k_2 \leq n$, and h_1, h_2, where $h_1 < h_2 \leq D_{i,j}$ for $i_1 \leq i \leq i_2$ and $j_1 \leq j \leq j_2$, $rec_D(i_1, i_2, k_1, k_2, h_1, h_2)$ is the 3D rectangle with the corners $(i_1 - 1, k_1 - 1, h_l)$, $(i_1 - 1, k_2, h_l)$, $(i_2, k_1 - 1, h_l)$, (i_2, k_2, h_l), where $l = 1, 2$, lying within $his(D)$.
3. For the matrix D, r_D is the minimum number of 3D rectangles $rec_D(i_1, i_2, k_1, k_2, h_1, h_2)$ which form a partition of $his(D)$. Note that $r_D \leq n^2$.

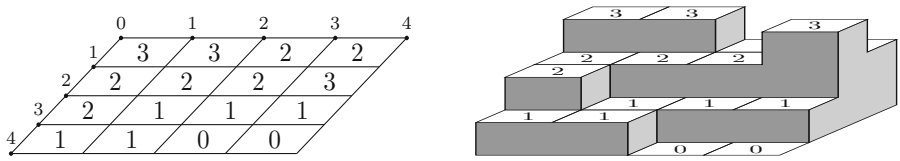


Fig. 1. (a) A matrix D on a grid, and (b) its corresponding histogram $his(D)$

3.2 Algorithms

Our first geometric algorithm for nonnegative integer matrix multiplication relies on the following key lemma.

Lemma 2. *Let P_A be a partition of the matrix A into 3D rectangles $rec_A(i_1, i_2, k_1, k_2, h_1, h_2)$, and let P_B be a partition of the matrix B into 3D rectangles $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2)$. For any $1 \leq i \leq n$, $1 \leq j \leq n$, the entry $C_{i,j}$ of the matrix product C of A and B is equal to the sum of $(h_2 - h_1)(h'_2 - h'_1) \times \#[k_1, k_2] \cap [k'_1, k'_2]$. over rectangle pairs $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A$, $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$ satisfying $i \in [i_1, i_2]$ and $j \in [j_1, j_2]$.*

Proof. For $1 \leq l_1 < l_2 \leq n$ and $1 \leq m_1 < m_2 \leq n$, let $I(l_1, l_2, m_1, m_2)$ be the $n \times n$ $0-1$ matrix where $I(l_1, l_2, m_1, m_2)_{i,k} = 1$ iff $l_1 \leq i \leq l_2$ and $m_1 \leq k \leq m_2$.

Clearly, we have $A = \sum_{rec_A(i_1,i_2,k_1,k_2,h_1,h_2) \in P_A} (h_2 - h_1) I(i_1, i_2, k_1, k_2)$. Similarly, we have $B = \sum_{rec_B(k'_1,k'_2,j_1,j_2,h'_1,h'_2) \in P_B} (h'_2 - h'_1) I(k'_1,' k_2, j_1, j_2)$.

It follows that $C = A \times B$ is the sum over pairs $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A$, $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$ of $(h_2 - h_1)(h'_1 - h'_2) I(i_1, i_2, k_1, k_2) \times I(k'_1, k'_2, j_1, j_2)$. It remains to observe that $(I(i_1, i_2, k_1 + 1, k_2) \times I(k'_1, k'_2, j_1 + 1, j_2))_{i,j} = \#[k_1, k_2] \cap [k'_1, k'_2]$ if $i_1 < i \leq i_2$ and $j_1 < j \leq j_2$ and it is equal to zero otherwise. □

Algorithm 1
Input: Two $n \times n$ matrices A, B with nonnegative integer entries.
Output: The arithmetic matrix product C of A and B.

1. Find a partition P_A of $his(A)$ into 3D rectangles $rec_A(i_1, i_2, k_1, k_2, h_1, h_2)$ whose number is within $O(1)$ of the minimum.
2. Find a partition P_B of $his(B)$ into 3D rectangles $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2)$ whose number is within $O(1)$ of the minimum.
3. Initialize an interval tree S on the k-coordinates of the rectangles in P_A and P_B. For each 3D rectangle $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A$ insert $[k_1, k_2]$, with a pointer to $A(i_1, i_2, k_1, k_2, h_1, h_2)$, into S.
4. Initialize interval lists $Start_j$, End_j, for $j = 1, \ldots, n$. For each rectangle $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$ report all intervals $[k_1, k_2]$ in S that intersect $[k'_1, k'_2]$. For each such interval $[k_1, k_2]$, with pointer to $rec_A(i_1, i_2, k_1, k_2, h_1, h_2)$, insert the interval $[i_1, i_2]$ with the weight $(h_2 - h_1) \times (h'_2 - h'_1) \times \#[k_1, k_2] \cap [k'_1, k'_2]$ into the lists $Start_{j_1}$ and End_{j_2}.

5. Initialize a weighted interval tree U on endpoints $1, \ldots, n$. For $j = 1, \ldots, n$, iterate the following steps. For $j > 1$, remove all weighted intervals $[i_1, i_2]$ on the list End_{j-1} from U. Insert all weighted intervals $[i_1, i_2]$ on the list $Start_j$ into U. For $i = 1, \ldots, n$, set $C_{i,j}$ to the value returned by U in response to the weight query at i.

Lemma 3. *Let $int(P_A, P_B)$ stand for the number of pairs $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A$, $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$, for which $[k_1, k_2] \cap [k'_1, k'_2] \neq \emptyset$. Algorithm 1 runs in time $\tilde{O}(n^2 + int(P_A, P_B)) = \tilde{O}(n^2 + r_A r_B)$.*

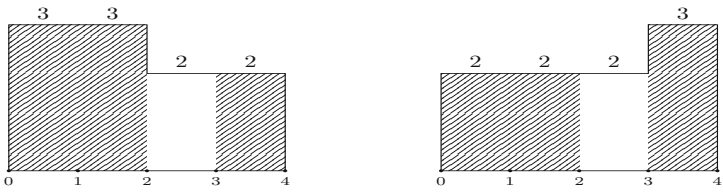
Proof. To implement steps 1 and 2 in $\tilde{O}(n^2)$ time, use the algorithm from the preceding section (Theorem 1). Step 3 can be implemented in $\tilde{O}(n+r_A+r_B) = O(n^2)$ time by Fact 1. In Step 4, the queries to S take $\tilde{O}(int(P_A, P_B))$ time by Fact 1. In Step 5, the initialization of the data structure U takes $\tilde{O}(n)$ time by Lemma 2. Next, the updates of the data structure U take $\tilde{O}(int(P_A, P_B))$ time by Lemma 2, while computing all columns of C takes $\tilde{O}(n^2)$ time by Remark 2. □

Theorem 2. *The matrix product of two $n \times n$ matrices A, B with nonnegative integer entries can be computed in $\tilde{O}(n^2 + r_A r_B)$ time.*

Proof. Algorithm 1 yields the theorem. Its correctness follows from Lemma 2 that basically says that for each pair of 3D rectangles, $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A$ and $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$, $C_{i,j}$ should be increased by $(h_2 - h_1) \times (h'_2 - h'_1) \times \#[k_1, k_2] \cap [k'_1, k'_2]$ for $i \in [i_1, i_2]$ and $j \in [j_1, j_2]$. In Step 4, two identical intervals $[i_1, i_2]$ corresponding to the left and right edge of the submatrix of C whose entries should be increased by the aforementioned value are inserted in the lists $Start_{j_1}$ and End_{j_2}, respectively. In both cases, they are weighted by the aforementioned value. In Step 5, in iteration j_1, the weighted interval $[i_1, i_2]$ from $Start_{j_1}$ is inserted into the weighted interval tree U, and in iteration $(j_2 + 1)$, it is removed from U as its copy is in End_{j_2}. In the iterations $j = j_1, \ldots, j_2$ in Step 5, when the interval $[i_1, i_2]$ is kept in the weighted interval tree, U and the entries of the submatrix $C_{i,j}$, $i_1 \leq i \leq i_2$, $j_1 \leq j \leq j_2$, are evaluated, the weight of the interval contributes to their value. The upper time bound follows from Lemma 3. □

When only one of the matrices A and B admits a partition of its 3D histogram into relatively few 3D rectangles and we have to assume the trivial partition of the other one into n^2 3D rectangles, the upper bound of Theorem 2 in terms of r_A, r_B and n seems too weak. In this case, an upper bound in terms of $int(P_A, P_B)$ and n in Lemma 3 may be much better. To derive a better upper bound in terms of just $\min\{r_A, r_B\}$ and n, we shall design another algorithm based on the slicing of the 3D histogram admitting a partition into relatively few 3D rectangles.

For an $n \times n$ matrix D with nonnegative integer entries and $i = 1, \ldots, n$, let $slice_i(D)$ stand for the part of $his(D)$ between the two planes perpendicular to the Y axis whose intersection with the XY plane are the horizontal lines $i - 1$ and i on the $[0, n] \times [0, n]$ grid. In other words, $slice_i(D)$ is a 3D histogram for



(a) **(b)**

Fig. 2. Let $slice_1(D)$ be the 2D histogram on the left and $slice_2(D)$ the 2D histogram on the right. Differentiating strips are shaded. Here, $gd(slice_1(D), slice_2(D)) = 2$.

the i-th row. Also note that a $slice_i(D)$ can be identified with a rectilinear 2D histogram; see Fig. 2 for an example. We define a *geometric distance* between two rectilinear 2D histograms H_1 and H_2 with a common base as the number of maximal vertical strips s such that:

1. for $i = 1, 2$, s contains exactly one maximal subsegment e_i of an edge of H_i different from and parallel to the base of the histograms, and
2. the subsegments e_1 and e_2 do not overlap.

See Fig. 2. We shall call such strips *differentiating strips*. For $slice_i(D)$ and $slice_k(D)$, we define the geometric distance $gd(slice_i(D), slice_k(D))$ as that for the corresponding rectilinear 2D histograms.

Lemma 4. *For an $n \times n$ matrix D with nonnegative integer entries, $\sum_{i=1}^{n-1} gd(slice_i(D), slice_{i+1}(D)) = O(r_D)$ holds.*

Proof. Each differentiating strip contributes, possibly jointly with one or two neighboring differentiating strips, to two vertices in the projected planar graph considered in the proof of Theorem 1. Thus, it contributes to the parameter m' in the aforementioned proof with at least 1. It follows $\sum_{i=1}^{n-1} gd(slice_i(D), slice_{i+1}(D)) \leq m'$. Hence, the inequality $m' \leq 4OPT$ established in the proof of Theorem 1 yields the thesis. □

Algorithm 2
Input: Two $n \times n$ matrices A and B with nonnegative integer entries.
Output: The matrix product C of A and B.

1. For $i = 1, \ldots, n-1$, find the differentiating strips for $slice_i(A)$ and $slice_{i+1}(A)$ and for each such strip s the indices $k_1(s)$ and $k_2(s)$ of the interval of entries $A_{i,k_1(s)}, \ldots, A_{i,k_2(s)}$ in the i-th row of A corresponding to it, as well as the difference $h(s)$ between the common value of each entry in $A_{i,k_1(s)}, \ldots, A_{i,k_2(s)}$ and the common value of each entry in $A_{i+1,k_1(s)}, \ldots, A_{i+1,k_2(s)}$.
2. For $j = 1, \ldots, n$, iterate the following steps:
   (a) Initialize a data structure T_j for counting partial sums of continuous fragments of the j-th column of the matrix B.
   (b) Compute $C_{1,j}$.
   (c) For $i = 1, \ldots, n-1$, iterate the following steps:

i. Set $C_{i+1,j}$ to $C_{i,j}$.
ii. For each differentiating strip s for $slice_i(A)$ and $slice_{i+1}(A)$, compute $\sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$ using T_j and set $C_{i+1,j}$ to $C_{i+1,j} + h(s) \sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$.

Lemma 5. *Algorithm 2 runs in $\tilde{O}(n(n + r_A))$ time.*

Proof. Step 1 can be easily implemented in $O(n^2)$ time. Step 2 (a) takes $\tilde{O}(n)$ time according to Fact 2 while Step 2 (b) can be trivially implemented in $O(n)$ time. Finally, based on Step 1, Step 2 (c) (ii) takes $\tilde{O}(gd(slice_i(D), slice_{i+1}(D))$ time. It follows that Step 2 (c) can be implemented in $\tilde{O}(\sum_{i=1}^{n-1} gd(slice_i(A), slice_{i+1}(A)))$ time, i.e., in $\tilde{O}(r_A)$ time by Lemma 4. Consequently, Step 2 takes $\tilde{O}(n(n + r_A))$ time. □

Theorem 3. *The arithmetic matrix product of two $n \times n$ matrices A, B with nonnegative integer entries can be computed in $\tilde{O}(n(n + \min\{r_A, r_B\}))$ time.*

Proof. The correctness of Algorithm 2 follows from the observation that a differentiating strip s for $slice_i(A)$ and $slice_{i+1}(A)$ yields the difference $h(s) \sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$ between $C_{i+1,j}$ and $C_{i,j}$ just on the fragment corresponding to $A_{i,k_1(s)}, \ldots, A_{i,k_2(s)}$ and $A_{i+1,k_1(s)}, \ldots, A_{i+1,k_2(s)}$, respectively. Lemma 5 yields the upper bound $\tilde{O}(n(n + r_A))$. The symmetric one $\tilde{O}(n(n + r_B))$ follows from the equalities $AB = (B^T A^T)^T$, $his(B) \equiv his(B^T)$, and consequently $r_B = r_{B^T}$. □

In Algorithm 2, the linear order in which the $C_{i,j}$ are updated to $C_{i+1,j}$ for $i = 1, \ldots, n-1$, along the row order of the matrix A is not necessarily optimal. Following the Boolean case [3,5], it may be more efficient to update $C_{i,j}$ while traversing a minimum spanning tree for the slices of $his(A)$ under the geometric distance. Here, however, we encounter the difficulty of constructing such an optimal spanning tree or a close approximation in substantially subcubic time. The next lemma will be useful.

Lemma 6. *Consider the family of rectilinear planar histograms with the base $[0, n]$, $n \geq 2$ and integer coordinates of its vertices in $[0, 2^M - 2]$, $M = O(\log n)$. There is a simple $O(n)$-time transformation of any histogram H in the family into an $0-1$ string $t(H)$, such that for any H_1 and H_2 in the family $gd(H_1, H_2) \leq ch(t(H_1), t(H_2)) \leq M gd(H_1, H_2)$, where $ch(\ ,\ )$ stands for the Hamming distance.*

Proof. Any histogram H in the family is uniquely represented by the vector $(H[1], \ldots, H[n]) \in \{1, \ldots, 2^M - 1\}^n$, where $H[1], \ldots, H[n]$ are the values of Y coordinates of the points on the "roof" of H increased by one with X coordinates $0.5, 1.5, \ldots, n - 0.5$ respectively.

For any $y \in \{0, \ldots, 2^M - 1\}$ denote its binary representation of length exactly M (padded with leading zeros if necessary) as $\mathrm{bin}(y)$.

Let $f(H, i) = \begin{cases} \mathrm{bin}(H[i]), & i = 1 \vee i > 1 \wedge H[i] \neq H[i-1] \\ \mathrm{bin}(0), & \text{otherwise.} \end{cases}$

The transformation t is then defined as $t(H) = f(H,1)\dots f(H,n)$. We have $ch(t(H_1), t(H_2)) = \sum_{i=1}^{n} ch(f(H_1,i), f(H_2,i))$ and

$$gd(H_1,H_2) = \begin{cases} 1, & H_1[1] \neq H_2[1] \\ 0, & \text{otherwise} \end{cases} + \\ + \sum_{i=2}^{n} \begin{cases} 1, & (H_1[i] \neq H_1[i-1] \vee H_2[i] \neq H_2[i-1]) \wedge (H_1[i] \neq H_2[i]) \\ 0, & \text{otherwise.} \end{cases}$$

Consider all possibilities that contribute exactly one to $gd(H_1, H_2)$:

1. $H_1[1] \neq H_2[1]$. In this case $f(H_1,1) = \text{bin}(H_1[1])$, $f(H_2,1) = \text{bin}(H_2[1])$ and $0 \leq ch(\text{bin}(H_1[1]), \text{bin}(H_2[1])) \leq M$.
2. $2 \leq i \leq n \wedge H_1[i] \neq H_1[i-1] \wedge H_2[i] = H_2[i-1] \wedge H_1[i] \neq H_2[i]$. In this case $f(H_1,i) = \text{bin}(H_1[i])$, $f(H_2,i) = \text{bin}(0)$ and $1 \leq ch(\text{bin}(H_1[i]), \text{bin}(0)) \leq M$.
3. $2 \leq i \leq n \wedge H_1[i] = H_1[i-1] \wedge H_2[i] \neq H_2[i-1] \wedge H_1[i] \neq H_2[i]$. See case 2.
4. $2 \leq i \leq n \wedge H_1[i] \neq H_1[i-1] \wedge H_2[i] \neq H_2[i-1] \wedge H_1[i] \neq H_2[i]$. See case 1.

To complete the proof, observe that in all other cases $ch(f(H_1,i), f(H_2,i)) = 0$. □

Fact 3 [6]. *For $\epsilon > 0$, a $(1+\epsilon)$-approximation minimum spanning tree for a set of n points in R^d with integer coordinates in $O(1)$ under the L_1 or L_2 metric can be computed by a Monte Carlo algorithm in $O(dn^{1+1/(1+\epsilon)})$ time.*

By combining the transformation of Lemma 6 with Fact 3 applied to the L_1 metric in $\{0,1\}^n$ and selecting $\epsilon = \log n$, we obtain a Monte Carlo $O(\log^2 n)$-approximation algorithm for the minimum spanning tree of the slices of $his(A)$ under the geometric distance, which runs in $\tilde{O}(n^2)$ time. This yields a generalization of Algorithm 2 to Algorithm 3, described in the full version of our paper. By an analysis of Algorithm 3 analogous to that of Algorithm 2 and a proof analogous to that of Theorem 3, we obtain a randomized generalization of Theorem 3:

Theorem 4. *Let A, B be two $n \times n$ matrices A, B with nonnegative integer entries in $[0, n^{O(1)}]$. Next, for $D \in \{A, B^T\}$, let M_D be the minimum cost of a spanning tree of slice$_i(D)$ for $i = 1,\dots,n$. The arithmetic matrix product of A and B can be computed by a randomized algorithm in $\tilde{O}(n(n+\min\{M_A,\ M_{B^T}\}))$ time with high probability.*

4 Final Remarks

A natural question is: Would it help to apply an algorithm that optimally rectangulates the 2D projection in Section 2.2? Although it would yield improved results in certain cases, it would not give a better approximation factor than 4 in general for the minimum rectangular 3D partition. An example of this is when the optimal 3D partition consists of k cubes lying on top of each other.

Then the 2D projection is k concentric squares of different sizes and an optimal rectangulation of the corresponding 2D projection consists of $4k-3$ rectangles. Hence, for large k, the approximation factor tends to 4.

The 4-approximation algorithm for minimum rectangular partition of a 3D histogram in case the histogram is $his(D)$ for an input $n \times n$ matrix D with nonnegative integer entries can easily be implemented in $O(n^2)$ time. Also note that the resulting partition of $his(D)$ can be used to form a compressed representation of D requiring solely $\tilde{O}(r_D)$ bits if the values of the entries in D are $n^{O(1)}$-bounded.

Our geometric algorithms for integer matrix multiplication can also be applied to derive faster $(1+\epsilon)$-approximation algorithms for integer matrix multiplication; if the range of an input matrix D is $[0, n^{O(1)}]$, then round each entry to the smallest integer power of $(1+\epsilon)$ that is not less than the entry. The resulting matrix D' has only a logarithmic number of different entry values and hence $r_{D'}$ may be much less than r_D.

Our algorithms and upper time bounds for integer $n \times n$ matrix multiplication can easily be extended to include integer rectangular matrix multiplication.

References

1. Bansal, N., Williams, R.: Regularity Lemmas and Combinatorial Algorithms. Theory of Computing **8**(1), 69–94 (2012)
2. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. 3rd edn. Springer, Santa Clara (2008)
3. Björklund, A., Lingas, A.: Fast boolean matrix multiplication for highly clustered data. In: Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 2001. LNCS, vol. 2125, p. 258. Springer, Heidelberg (2001)
4. Dielissen, V.J., Kaldewai, A.: Rectangular Partition is Polynomial in Two Dimensions but NP-Complete in Three. Information Processing Letters **38**(1), 1–6 (1991)
5. Gasieniec, L., Lingas, A.: An Improved Bound on Boolean Matrix Multiplication for Highly Clustered Data. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 329–339. Springer, Heidelberg (2003)
6. Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: Proc. of STOC 1998, pp. 604–613 (1998)
7. Keil, J.M.: Polygon Decomposition. Survey, Dept. Comput. Sc. Univ. Saskatchewan (1996)
8. Le Gall, F.: Powers of Tensors and Fast Matrix Multiplication. In: Proc. of the 39th ISSAC, pp. 296–303 (2014)
9. Lingas, A.: A Geometric Approach to Boolean Matrix Multiplication. In: Bose, P., Morin, P. (eds.) ISAAC 2002. LNCS, vol. 2518, pp. 501–510. Springer, Heidelberg (2002)
10. Lipski, W.: Finding a Manhattan path and related problems. Networks **13**(3), 399–409 (1983)

11. Mehlhorn, K.: Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry. EATCS Monographs on Theo. Comput. Sc., Springer (1984)
12. Muthukrishnan, S., Poosala, V., Suel, T.: On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 236–256. Springer, Heidelberg (1998)
13. Sack, J.-R., Urrutia, J. (ed).: Handbook of Computational Geometry. Elsevier (2000)