Journal of Supercomputing, 15, 51–93 (2000)

Image-Space Decomposition Algorithms for Sort-First Parallel Volume Rendering of Unstructured Grids*

HÜSEYIN KUTLUCA kutluca@cs.bilkent.edu.tr
Computer Engineering Department, Bilkent University, TR-06533 Ankara, Turkey

TAHSIN M. KURÇ kurc@cs.umd.edu
Computer Science Department, University of Maryland, College Park, MD 20742

CEVDET AYKANAT aykanat@cs.bilkent.edu.tr
Computer Engineering Department, Bilkent University, TR-06533 Ankara, Turkey

(Received February 26, 1998; final version accepted November 17, 1998)

Abstract. Twelve adaptive image-space decomposition algorithms are presented for sort-first parallel direct volume rendering (DVR) of unstructured grids on distributed-memory architectures. The algorithms are presented under a novel taxonomy based on the dimension of the screen decomposition, the dimension of the workload arrays used in the decomposition, and the scheme used for workload-array creation and querying the workload of a region. For the 2D decomposition schemes using 2D workload arrays, a novel scheme is proposed to query the exact number of screen-space bounding boxes of the primitives in a screen region in constant time. A probe-based chains-on-chains partitioning algorithm is exploited for load balancing in optimal 1D decomposition and iterative 2D rectilinear decomposition (RD). A new probe-based optimal 2D jagged decomposition (OJD) is proposed which is much faster than the dynamic-programming based OJD scheme proposed in the literature. The summed-area table is successfully exploited to query the workload of a rectangular region in constant time in both OJD and RD schemes for the subdivision of general 2D workload arrays. Two orthogonal recursive bisection (ORB) variants are adapted to relax the straight-line division restriction in conventional ORB through using the medians-of-medians approach on regular mesh and quadtree superimposed on the screen. Two approaches based on the Hilbert space-filling curve and graph-partitioning are also proposed. An efficient primitive classification scheme is proposed for redistribution in 1D, and 2D rectilinear and jagged decompositions. The performance comparison of the decomposition algorithms is modeled by establishing appropriate quality measures for load-balancing, amount of primitive replication and parallel execution time. The experimental results on a Parsytec CC system using a set of benchmark volumetric datasets verify the validity of the proposed performance models. The performance evaluation of the decomposition algorithms is also carried out through the sort-first parallelization of an efficient DVR algorithm.

Keywords: direct volume rendering, unstructured grids, sort-first parallelism, image-space decomposition, chains-on-chains partitioning, jagged decomposition, rectilinear decomposition, orthogonal recursive bisection, Hilbert space-filling curve, graph partitioning

* This work is partially supported by the Commission of the European Communities, Directorate General for Industry under contract ITDC 204-82166, and Turkish Science and Research Council under grant EEEAG-160

1. Introduction

In many fields of science and engineering, computer simulations provide a cheap and controlled way of investigating physical phenomena. The output of these simulations is usually a large amount of numerical values, which makes it very difficult for scientists to extract useful information from the data to derive useful conclusions. Therefore, visualizing large quantities of numerical data as an image provides an indispensable tool for researchers. In many engineering simulations, datasets consist of numerical values which are obtained at points (sample points) distributed in a volume that represents the physical environment. The sample points constitute a *volumetric grid* superimposed on the volume, and they are connected to some other nearby sample points to form volume elements (*cells*). Volumetric grids can be divided into two categories as: *structured* and *unstructured*. Structured grids are topologically equivalent to the integer lattices, and as such, can easily be represented by a 3D array. In unstructured grids, the sample points in the volume data are distributed irregularly over 3 dimensional (3D) space and there may be voids in the volumetric grid. Unstructured grids are also called *cell oriented* grids, because they are represented as a list of cells with pointers to the sample points that form the respective cells. With recent advances in generating higher quality adaptive meshes, unstructured grids are becoming increasingly popular in the simulation of scientific and engineering problems with complex geometries.

There are two major categories of volume rendering methods: *indirect* and *direct*. Indirect methods extract intermediate geometrical representation of the data (i.e. isosurfaces), and render those surfaces via conventional surface rendering methods. Direct methods render the data without generating an intermediate representation, hence they are more general, flexible and have the potential to provide more complete information about the data being visualized. However, direct methods are slow due to massive computations. Large scale scientific and engineering simulations are usually performed in parallel on distributed-memory architectures. Parallel visualization of the vast amount of volumetric data produced by these simulations on the same parallel machine saves the time to transfer the data from the parallel machine to a sequential graphics workstation over possibly slow communication links. Hence, direct volume rendering (DVR) is a good candidate for parallelization on distributed-memory multicomputers.

1.1. Direct volume rendering (DVR)

In general, the DVR methods consist of two main phases. These are *resampling* and *composition* phases, which are in general manipulated in a highly interleaved manner. In the resampling phase, new samples are interpolated by using the original sample points. The rendering method should somehow locate the new sample point in the cell domain, because the vertices of that cell, in which the new sample is being generated, will be used in the interpolation for the new sample. This problem is known as the *point location* problem. In the composition phase, the new samples generated are mapped to color and opacity values, which are composited to

determine the contribution of the data on a pixel. The composition operation is associative, but not commutative; therefore these color and opacity values should be composited in visibility order. The determination of the correct composition order is known as the *view sort* problem.

The DVR algorithms for unstructured grids can be classified into three categories: *ray-casting*, *projection* and *hybrid* [31]. In the ray-casting methods, the image space is traversed to cast a ray for each pixel, and each ray is followed, sampled and composited along the volume. In the projection methods, the volume is traversed to perform a view-dependent depth sort on the cells. Then, all cells are projected onto the screen, in this sorted order, to find their contributions to the image and composite them. In the hybrid methods, the volume is traversed in object order such that the contributions of the cells to the image are accumulated in image order.

A DVR application contains two interacting domains: object space and image space. The object space is a 3D domain containing the volume data to be visualized. The image space (screen) is a 2D domain containing pixels from which rays are shot into the 3D object domain to determine the color values of the respective pixels. Based on these domains, there are basically two approaches for data parallel DVR: object-space parallelism and image-space parallelism. The cells or cell clusters constitute the atomic tasks in the object-space parallelism, whereas the pixels or pixel blocks constitute the atomic tasks in the image-space parallelism. The parallel DVR algorithms can also be categorized according to the taxonomy proposed by Molnar et. al. [18] for parallel polygon rendering. In polygon rendering, the rendering process is a pipeline of operations applied to the primitives in the scene. This rendering pipeline has two major steps called *geometry processing* and *rasterization*. Molnar et al. [18] provides a classification of parallelism, based on the point of data redistribution step in the rendering pipeline, as *sort-first* (before geometry processing), *sort-middle* (between geometry processing and rasterization), and *sort-last* (after rasterization). In this taxonomy, the image-space and object-space parallelism in DVR can be considered as the sort-first and sort-last parallelism, respectively.

In sort-last (object-space) parallel DVR, the 3D object domain is decomposed into disjoint subvolumes and each subvolume is concurrently rendered by a distinct processor. At the end of this local rendering phase, incomplete full-screen images are created at each processor. In the pixel-merging phase, these local partial images are merged for composition over the interconnection network. The sort-last parallelism is a promising approach offering excellent scalability in terms of the number of primitives it can handle. However, it suffers from high interprocessor communication volume during the pixel merging phase especially at high screen resolutions.

1.2. Sort-first parallel DVR

In sort-first (image-space) parallel DVR, which is the focus of this work, each processor is initially assigned a subset of primitives in the scene. The screen is decomposed into regions and each region is assigned to a distinct processor for rendering. The primitives are then redistributed among the processors so that each processor has all the primitives whose projection areas intersect the region assigned to it. The

primitives whose projection areas intersect more than one region are replicated in the processors assigned to those regions. After primitive redistribution, each processor performs local rendering operations on its region. The sort-first parallelism is a promising approach since each processor generates a complete image for its local screen subregion. However, it faces load-balancing problems in the DVR of unstructured grids due to uneven on-screen primitive distribution. Hence, image-space decomposition is a crucial factor in the performance of the sort-first parallelism.

The image-space decomposition schemes for sort-first parallel DVR can be classified as *static*, *dynamic* and *adaptive*. Static decomposition is a view-independent scheme and the load-balancing problem is solved implicitly by the scattered assignment of the pixels or pixel blocks. The load-balancing performance of this scheme depends on the assumption that neighbor pixels are likely to have equal workload since they are likely to have similar views of the volume. The scattered assignment scheme has the advantage that assignment of screen regions to processors is known a priori and static irrespective of the data. However, since the scattered assignment scheme assigns adjacent pixels or pixel blocks to different processors, it disturbs the image-space coherency and increases the amount of primitive replication. Here, the image-space coherency relies on the observation that rays shot from nearby pixels are likely to pass through the same cells involving similar computations. In addition, since decomposition is done irrespective of input data, it is still possible that some regions of the screen are heavily loaded and some processors may perform substantially more work than the others. In the dynamic approach, pixel blocks are assigned to processors in a demand-driven basis when they become idle. The dynamic approach also suffers from disturbing the image-space coherency since adjacent pixel blocks may be processed by different processors. Furthermore, since region assignments are not known a priori, each assignment should be broadcast to all processors so that necessary primitive data is transmitted to the respective processor. Hence, the dynamic scheme is not a viable approach for message-passing distributed memory architectures. Adaptive decomposition is a view-dependent scheme and the load-balancing problem is solved explicitly by using the primitive distribution on the screen. The adaptive scheme is a promising approach since it handles the load-balancing problem explicitly and it preserves the image-space coherency as much as possible by assigning contiguous pixels to the processors.

1.3. Previous work

Most of the previous work on parallel DVR of unstructured grids evolved on shared-memory multicomputers [3, 5, 29]. Challinger [3, 5] presents image-space parallelization of a hybrid DVR algorithm [4] for BBN TC2000 shared-memory multicomputer. In the former work [3], the scanlines are considered as the atomic tasks and they are assigned to the processors using the static (scattered) and dynamic (demand-driven) schemes for two different algorithms. In the latter work [5], the screen is divided into square pixel blocks which are considered as the atomic tasks for dynamic assignment. The pixel blocks are sorted in decreasing order according to the number of primitives associated with them, and they are considered in this

sorted order for dynamic assignment to achieve better load-balancing. Williams [29] presents object-space parallelization of a projection based DVR algorithm [30] on Silicon Graphics Power Series. The target machine is a shared-memory multicomputer with computer graphics enhancement. Ma [15] investigates sort-last parallelization of a ray-casting based DVR algorithm [8] on an Intel Paragon multicomputer which is a message-passing distributed-memory architecture.

1.4. Contributions

In this work, we investigate sort-first parallelism for DVR of unstructured grids on distributed-memory architectures. This type of parallelism was not previously utilized in DVR of unstructured grids on distributed-memory multicomputers. We present twelve adaptive image-space decomposition algorithms under a novel taxonomy. The proposed taxonomy is based on the dimension of the screen decomposition and the dimension of the workload arrays used in the decomposition. The decomposition algorithms are parallelized as much as possible to reduce the preprocessing overhead. Table 1 displays the acronyms used for the image-space decomposition algorithms listed according to the classification of the proposed taxonomy.

As in the previous works on parallel polygon rendering [7, 20, 25, 28], the number of primitives that fall onto a region is used to represent the workload of the region. The screen-space bounding-box of the projection area of a primitive is used to approximate the coverage of the primitive on the screen. The bounding-box approximation is selected to avoid expensive computations needed in finding the ex-

Table 1. The image-space decomposition algorithms

Classification	Abbreviation	Complete name	Primitive classification
	HHD	Heuristic Horizontal Decomposition	Inverse-mapping
1D-1D-exact	OHD	Optimal Horizontal Decomposition	Inverse mapping
	HJD	Heuristic Jagged Decomposition	Inverse mapping
2D-1D-exact	ORB-1D	Orthogonal Recursive Bisection with 1D arrays	Rectangle-intersection
	OJD-I	Optimal Jagged Decomposition using Inverse area heuristic for workload array	Inverse mapping
	ORB-I	Orthogonal Recursive Bisection using Inverse area heuristic for workload array	Rectangle-intersection
	ORBMM-Q	Orthogonal Recursive Bisection with Medians of Medians on Quadtree	Mesh
2D-2D-IAH	ORBMM-M	Orthogonal Recursive Bisection with Medians of Medians on cartesian Mesh	Mesh
	HCD	Hilbert Curve based Decomposition	Mesh
	GPD	Graph Partitioning based Decomposition	Mesh
	OJD-E	Optimal Jagged Decomposition using Exact model for workload array	Inverse mapping
2D-2D-exact	RD	Rectilinear Decomposition	Inverse mapping

act primitive coverage. All the decomposition algorithms query the *bounding-box counts* (bb-counts) of the screen subregions repeatedly during the subdivision process. Here, the bb-count of a region refers to the number of bounding boxes intersecting the region. We show that the exact bb-count of a stripe along one of the dimensions can be found in constant time using two 1D workload arrays. This idea is also exploited in some of the 2D decomposition algorithms producing rectangular subregions in a hierarchical manner through using 1D workload arrays. In a previous work by Muller [20], a heuristic scheme, referred to here as the *inverse area heuristic* (IAH), is used to estimate the bb-count of a region using one 2D array. In this work, we propose a novel scheme, referred to here as the *exact* model, to query the exact bb-count of a rectangular region in constant time by using four 2D workload arrays. Both IAH and exact models are used in the 2D decomposition algorithms utilizing 2D workload arrays.

We present subdivision algorithms that find optimal decompositions for load-balancing. The load-balancing problem in 1D decomposition is modeled as the well-known *chains-on-chains partitioning* (CCP) problem [2]. The objective in the CCP problem is to divide a given chain of modules into a number of consecutive subchains such that the load of the most heavily loaded subchain is minimized. An efficient probe-based CCP algorithm is utilized for optimal 1D (horizontal) decomposition. The probe-based CCP algorithm is also exploited for two different 2D decomposition schemes: optimal jagged decomposition (OJD) and heuristic rectilinear decomposition (RD). The proposed OJD scheme is a novel scheme applicable to the subdivision of general 2D workload arrays, and it is much faster than the dynamic-programming based OJD scheme proposed in the literature [17]. The summed-area table [6] is successfully exploited to query the workload of a rectangular region in constant time in both OJD and RD schemes for the subdivision of general 2D workload arrays.

Three distinct 2D decomposition approaches, which generate non-rectangular regions, are also investigated for image-space decomposition. Two orthogonal recursive bisection (ORB) variants are implemented to alleviate the load-imbalance problem due to the straight-line division restriction in the ORB scheme adopted by Mueller [20] (referred to as MAHD in his work). These ORB variants apply the *medians-of-medians* [23, 27] approach on regular mesh and quadtree superimposed on the screen. The Hilbert space-filling curve [19, 23, 27] is exploited for image-space decomposition. Finally, image-space decomposition is modeled as a graph-partitioning problem and state-of-the-art graph partitioning tool MeTiS [10] is used for partitioning the generated graph.

Three classification schemes are investigated for primitive redistribution to be performed according to the region-to-processor assignment. A straightforward mesh-based local primitive classification algorithm can be used for the decomposition algorithms that generate both rectangular and non-rectangular regions. However, it is computationally very expensive since it involves tallying the bounding boxes of the primitives. The rectangle-intersection based classification algorithm is more efficient for the decomposition schemes that generate rectangular regions. Furthermore, a much more efficient inverse-mapping based classification scheme is proposed for the 1D horizontal, 2D jagged and 2D rectilinear decomposition

schemes. The proposed scheme exploits the regular structure of the decompositions produced by these schemes so that its complexity per primitive is independent of both the number of processors and the screen resolution.

The load-balancing, primitive-replication and parallel run-time performances of the decomposition algorithms are compared both theoretically and experimentally. The theoretical models for the comparison of load-balancing and primitive-replication performances are based on establishing appropriate quality measures. The experimental results on a Parsytec CC system using a set of benchmark volumetric datasets verify the validity of the proposed quality measures. The performance evaluation of the presented image-space decomposition algorithms is also carried out through sort-first parallelization of Challinger's [4] hybrid DVR algorithm on a Parsytec CC system.

The rest of the paper is organized as follows. Section 2 presents Challinger's DVR algorithm [4] and describes its sort-first parallelization. Section 3 summarizes the basic steps involved in the parallel image-space decomposition. The taxonomy of the image-space decomposition algorithms is introduced in Section 4. Section 5 presents the workload-array creation algorithms which enable the efficient query of the workloads associated with the regions during the subdivision phase. The image-space decomposition algorithms are presented and discussed in Section 6. The primitive classification algorithms for the redistribution phase are given in Section 7. The models for the theoretical performance comparison of the decomposition algorithms are presented in Section 8. Finally, Section 9 presents the experimental results.

2. DVR algorithm

The sequential rendering algorithm chosen for sort-first parallelization is based on the algorithm developed by Challinger [4]. This algorithm adopts the basic ideas in the scanline Z-buffer based polygon rendering algorithms to resolve the point location and view sort problems. As a result, the algorithm requires that the volumetric dataset is composed of cells with planar faces. This algorithm does not require connectivity information between cells, and it has the power to handle non-convex and cyclic grids. In this work, the volumetric dataset is assumed to be composed of tetrahedral cells. If a dataset contains volume elements that are not tetrahedral, these elements can be converted into tetrahedral cells by subdividing them [8, 26]. A tetrahedral cell has four points and each face of a tetrahedral cell is a triangle, thus easily meeting the requirement of cells with planar faces. Since the algorithm operates on polygons, the tetrahedral dataset is further converted into a set of distinct triangles. Only triangle information is stored in the data files.

The algorithm starts with an initialization phase which involves the sorting of the triangular faces (primitives) into a *y-bucket* structure according to their minimum *y*-values in the screen coordinates. Each entry of the *y*-bucket corresponds to a scanline on the screen. The algorithm performs the following initialization operations for each successive scanline on the screen. *Active primitive* and *active edge* lists for the current scanline are updated incrementally. The active primitive list stores the triangles intersecting the current scanline, and it is updated incrementally using

the y-bucket. The active edge list stores the triangle edges intersecting the current scanline. Finally, a span is generated for the active edge pair of each triangle in the active primitive list, and it is sorted into an x-bucket according to its minimum x value. Each entry of the x-bucket corresponds to a pixel location on the current scanline.

Each successive pixel on a scanline is processed as follows. An *active span-list* is maintained and updated incrementally along the scanline. Each entry of the active span-list stores the z coordinate of the intersection point (z-intersection) of the primitive that generates the span with the ray shot from the respective pixel location, span information, a pointer to the primitive, and a flag to indicate whether the primitive is an exterior or an interior face of the volume. If a face of a cell is shared by two cells, that face is called interior. If it is not shared by any other cell, the face is called exterior. The spans in the active span-list are maintained in sorted increasing order according to their z-intersection values. The z-intersection values are calculated by the incremental rasterization of the spans stored in the x-bucket. Two consecutive spans in the list, if at least one of them belongs to an interior triangle, correspond to the entry and exit faces of a tetrahedral cell hit by the respective ray in the visibility order. As the entry and exit point z values are known from the z-intersection values, a new sample is generated in the middle of the line segment, which is formed by the entry and exit points of the respective ray intersecting the cell, through 3D inverse-distance interpolation of the scalar values at the corner points of the respective triangles. The color and opacity values computed through mid-point sampling along the ray within the cell are composited to the pixel.

The algorithm exploits image-space coherency for efficiency. The calculations of intersections of triangular faces with the scanline, and insertion and deletion operations on the active primitive list are done incrementally. This type of coherency is called *inter-scanline* coherency. The z-intersection calculations through span rasterization, depth-sorting of active spans according to their z-intersection values, insertion to and deletion from the active span-list are done incrementally. This type of coherency is called *intra-scanline* coherency.

2.1. Workload model

As mentioned earlier, the number of primitives that falls onto a region is used to approximate the workload of the region through bounding-box approximation. This workload model is used in the presented image-space decomposition algorithms for the sake of general performance evaluation of sort-first parallel DVR. However, there are three parameters affecting the computational load of a screen region in Challinger's DVR algorithm [4]. The first one is the number of triangles (primitives), because the total workload of a region due to the transformation of triangles, insertion operations into the y-bucket, and insertions into and deletions from the active primitive list is proportional to the number of triangles in the region. The second parameter is the number of scanlines each triangle extends. This parameter represents the computational workload associated with the construction of the

edge intersections (hence, corresponding spans), edge rasterization, clipping spans to the region boundaries, and insertion of the spans into the x-bucket. The total number of pixels generated through the rasterization of these spans is the third parameter affecting the computational load of a region. Each pixel generated adds computations required for sorting, insertions to and deletions from the active-span list, and interpolation and composition operations. Therefore, the workload (WL) in a region can be approximated as;

$$WL = \alpha_T N_T + \alpha_S N_S + \alpha_P N_P. \quad [1]$$

Here, N_T, N_S, and N_P represent the number of triangles, spans, and pixels, respectively, to be processed in a region. In Eq. 1, α_T, α_S, and α_P represent the relative unit computational costs of the operations associated with triangles, spans, and pixels, respectively.

Finding the exact number of spans and pixels generated in a region due to a triangle requires the rasterization of the triangle. The bounding-box approximation used in finding the bb-count of a region is also exploited for estimating the span and pixel counts of the region. That is, a triangle whose bounding box has corner points $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$ is assumed to generate $y_{\max} - y_{\min} + 1$ spans and $(y_{\max} - y_{\min} + 1) \times (x_{\max} - x_{\min} + 1)$ pixels. It is clear that the bounding-box approximation incurs errors in the estimation of the pixel count of a region in both 1D and 2D decompositions. The bounding-box approximation may also incur errors in the estimation of the span count of a region in 2D decompositions. For example, the triangle shown in Figure 1 contributes only one span to the span count of region R_2, while contributing four spans to the span counts of the other three regions R_1, R_3 and R_4. However, the bounding-box approximation incurs four spans for each one of the four regions, thus inducing error in the estimation of the span count of region R_2.

It should be noted here that only the bb-counts are considered in the workload model in the discussion of the image-space decomposition algorithms for the sake of clarity of the presentation. The pixel and span counts can easily be incorporated into the workload model by treating each span and pixel covered by the bounding box of the triangle as bounding boxes of proper height and width with computational loads of α_S and α_P, respectively. As will be discussed in Section 9, incorporating the span and pixel counts into the workload model substantially increases the performance of the parallelization of Challinger's DVR algorithm [4] through better load balancing.

3. Parallel image-space decomposition

Parallel adaptive image-space decomposition on distributed-memory architectures involves five consecutive phases: bounding-box creation, local workload-array creation, global-sum operation on local workload arrays, subdivision and primitive redistribution. In the bounding-box creation phase, processors concurrently create the screen-space bounding boxes of their local primitives. In the workload-array creation phase, processors concurrently create their local workload arrays using the

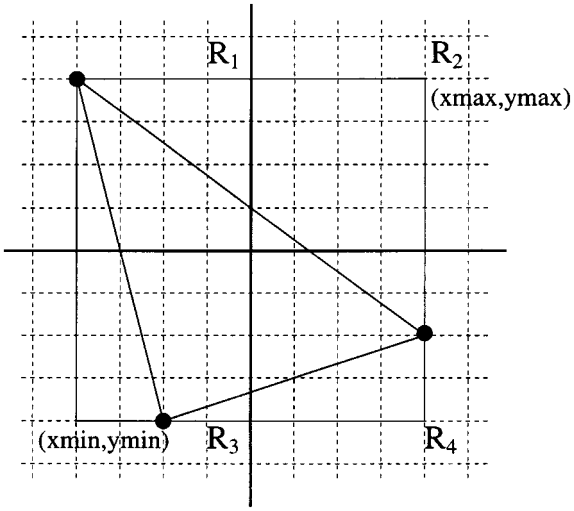


Figure 1. The bounding-box approximation.

distribution of their local bounding boxes on the full screen. Then, a global-sum operation is performed on the local arrays so that all processors receive the copies of the global workload array(s). In the subdivision phase, the screen is subdivided into P regions using the workload arrays so that each processor is assigned to a single region. In some of the decomposition schemes, the subdivision operation is also performed in parallel, whereas in the other schemes the subdivision operation is not parallelized because of either the sequential nature or the fine granularity of the subdivision algorithm. The schemes adopting parallel subdivision necessitate a final all-to-all broadcast operation so that each processor gets all the region-to-processor assignments. In the redistribution phase, processors concurrently classify their local primitives according to the region-to-processor assignments. Then, the primitives concurrently migrate to their new home processors through an all-to-all personalized communication operation.

4. A taxonomy of the decomposition algorithms

The taxonomy (see Figure 2) proposed for the image-space decomposition algorithms is based on the decomposition strategy and workload arrays used in the decomposition. The first classification is based on the dimension of the decomposition of the screen, which is a 2D space. The *1D decomposition* algorithms divide in only one dimension of the screen. These algorithms utilize the workload distribution with respect to only one dimension. On the other hand, the *2D decomposition* algorithms divide the screen in two dimensions by utilizing the workload distribution with respect to both dimensions of the screen.

The second classification is based on the dimension of the workload arrays used in the decomposition. The term *arrays* will also be used to refer to *workload arrays*.

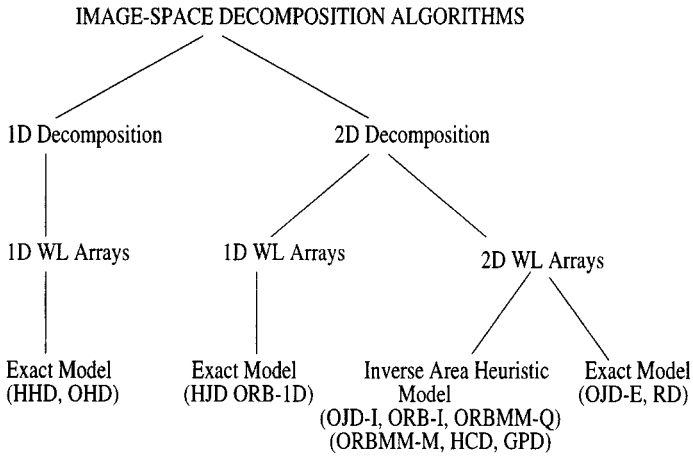


Figure 2. The taxonomy of the image-space decomposition algorithms. The abbreviations in the parentheses denote the algorithms in the respective class (see Table 1 for the complete names).

In the 1D arrays scheme, *1D arrays* are used to find the distribution of the workload along each dimension of the screen. In the 2D arrays scheme, a 2D coarse mesh is superimposed on the screen and the distribution of the workload over this mesh is used to divide the screen.

The third classification is based on the scheme used for workload-array creation and querying the workload of a region, especially for the 2D decomposition schemes utilizing 2D arrays. In the *inverse area heuristic* (IAH) model, an estimate of the bb-count of a region is found, whereas in the exact model, the exact bb-count of a region is found. The IAH model handles rectangular and non-rectangular regions as well as regions consisting of non-adjacent mesh cells when 2D arrays are used. However, the exact model is only used for rectangular regions because of the high computational cost of finding the exact bb-counts of non-rectangular and disconnected regions.

5. Workload-array creation

This section describes the workload-array creation algorithms for a screen of resolution $M \times N$. The main objective of the presented algorithms is to enable the efficient query of workloads associated with regions during the subdivision operation.

5.1. 1D arrays

Here, we present an algorithm which enables to query the exact bb-count of a stripe in $O(1)$ time by using two 1D arrays. Without loss of generality, we will restrict our discussion to 1D arrays for the y-dimension of the screen. The first array is the *y-dimension start* (YS) array of size M. The second array is the *y-dimension end* (YE)

array of size M. Each entry of these arrays corresponds to a scanline in the screen. Each bounding box with y-extent $(y_{\min}, y_{\max})$ increments YS$[y_{\min}]$ and YE$[y_{\max}]$ by one. After processing all bounding boxes, a prefix-sum operation is performed on both arrays. Hence, YS$[j]$ gives the number of bounding boxes that start before scanline j, including scanline j, whereas YE$[j]$ gives the number of bounding boxes that end before scanline j, including scanline j. The workload (WL)—in terms of the exact bb-count—of a horizontal stripe bounded by scanlines i and j ($> i$) is computed as:

$$WL[i, j] = YS[j] - YE[i - 1]. \quad [2]$$

5.2. 2D arrays

In the algorithms using 2D arrays, a 2D coarse mesh of size $m \times n$ is superimposed on the screen to make the decomposition affordable both in terms of space and execution time. The mesh cell weights are computed using the distribution of the bounding boxes over this coarse mesh.

5.2.1. Inverse-area-heuristic (IAH) model. In this scheme, the bounding boxes are tallied to the mesh cells. Some bounding boxes may intersect multiple cells. The IAH model [20] is utilized to decrease the amount of errors due to counting such bounding boxes many times. Each bounding box increments the weight of each cell it intersects by a value inversely proportional to the number of cells the bounding box intersects.

In the IAH model, if the screen regions generated by a decomposition algorithm do not have any shared bounding boxes then the sum of the weights of the cells forming each region gives the exact bb-count of that region. However, the shared bounding boxes still cause errors when calculating the bb-count of a region. The contribution of a bounding box shared among regions is divided among those regions. Thus, the computed workload of a region is less than the actual bb-count. However, the error is expected to be less than counting shared bounding boxes multiple times while adding cell weights.

In order to query the workload of a rectangular region in $O(1)$ time, 2D workload array T of size $m \times n$ is converted into a summed area table (SAT) [6] by performing a *2D prefix sum* over the T array. The 2D prefix sum is done by performing a 1D prefix sum on each individual row of T followed by a 1D prefix sum on each individual column of T. After the 2D prefix-sum operation, $T[x, y]$ gives the workload of the region $[(1, 1); (x, y)]$ bounded by the corner points $(1, 1)$ and (x, y). Hence, the workload of a rectangular region $[(x_{\min}, y_{\min}); (x_{\max}, y_{\max})]$ can be computed using the following expression;

$$\begin{aligned} WL = {} & T[x_{\max}, y_{\max}] - T[x_{\max}, y_{\min} - 1] - T[x_{\min} - 1, y_{\max}] \\ & + T[x_{\min} - 1, y_{\min} - 1]. \end{aligned} \quad [3]$$

5.2.2. Exact model. In this work, we propose an efficient algorithm for finding the exact number of bounding boxes in a rectangular region. The proposed method uses four 2D arrays SXY, EXY, EX, EY each of size $m \times n$ (see Figure 3(a)). Each bounding box with x-extent $(x_{\min}, x_{\max})$ and y-extent $(y_{\min}, y_{\max})$ increments both SXY$[x_{\min}, y_{\min}]$ and EXY$[x_{\max}, y_{\max}]$ by one. Each bounding box also increments both EX$[x_{\max}, y]$ and EY$[x, y_{\max}]$ by one for $y_{\min} \le y \le y_{\max}$ and $x_{\min} \le x \le x_{\max}$, respectively. These operations correspond to rasterizing the right and top sides of the bounding box for computing its contribution to the EX and EY arrays, respectively. After processing all bounding boxes, a 2D prefix-sum operation is performed on both SXY and EXY arrays. In addition, rowwise and columnwise 1D prefix-sum operations are performed on the EX and EY arrays, respectively. After the prefix-sum operations on the arrays, SXY$[x, y]$ gives the number of bounding boxes intersecting the region $[(1, 1); (x, y)]$. In other words, it finds the number of bounding boxes that start (or whose lower-left corners reside) in that region. EXY$[x, y]$ gives the number of bounding boxes that end (whose upper-right corners reside) in the region $[(1, 1); (x, y)]$. EX$[x, y]$ gives the number of bounding boxes whose right sides intersect the line segment $[(1, y), (x, y)]$. EY$[x, y]$ gives the number of bounding boxes whose top sides intersect the line $[(x, 1), (x, y)]$.

After defining the arrays SXY, EXY, EX and EY, we can calculate the exact bb-count (WL) of a region $[(x_{\min}, y_{\min}); (x_{\max}, y_{\max})]$ as follows;

$$\begin{aligned} WL = {} & \mathrm{SXY}[x_{\max}, y_{\max}] - \mathrm{EXY}[x_{\min} - 1, y_{\max} - 1] \\ & - \mathrm{EXY}[x_{\max} - 1, y_{\min} - 1] \\ & + \mathrm{EXY}[x_{\min} - 1, y_{\min} - 1] - \mathrm{EX}[x_{\min} - 1, y_{\max}] \\ & - \mathrm{EY}[x_{\max}, y_{\min} - 1]. \end{aligned} \quad [4]$$

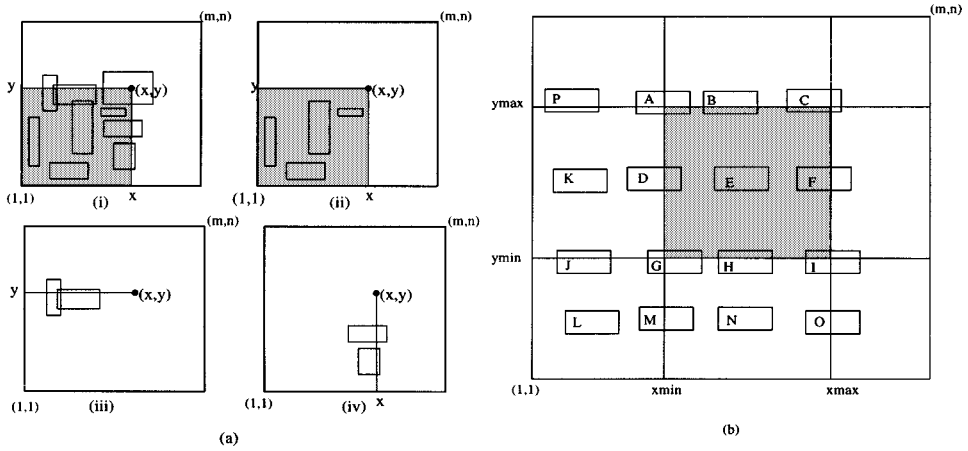


Figure 3. (a) The 2D arrays used for the exact model: (i) SXY, (ii) EXY, (iii) EX, and (iv) EY. (b) The exact model for calculating the exact number of bounding boxes in the shaded region.

As seen in Eq. 4, the proposed scheme requires only 4 subtractions and 1 addition to query the exact bb-count of a rectangular region. The correctness of Eq. 4 can easily be verified with the help of Fig. 3(b) as follows. Each letter $(A - P)$ in the figure represents the bounding boxes of the same type. The exact bb-count of the shaded rectangular region is equal to the number of bounding boxes of types $A - I$. $\mathrm{SXY}[x_{\max}, y_{\max}]$ gives the number of bounding boxes of types $A - P$. $\mathrm{EXY}[x_{\min} - 1, y_{\max} - 1]$ gives the number of bounding boxes of types J, K, L and $\mathrm{EXY}[x_{\max} - 1, y_{\min} - 1]$ gives the number of bounding boxes of types L, M, N. Since we subtract the number of L-type bounding boxes twice, we add $\mathrm{EXY}[x_{\min} - 1, y_{\min} - 1]$ to the sum. Arrays SXY and EXY are not sufficient for the exact computation because of the bounding boxes of types O and P. So, we use $\mathrm{EX}[x_{\min} - 1, y_{\max}]$ and $\mathrm{EY}[x_{\max}, y_{\min} - 1]$ to subtract the number of bounding boxes of types P and O, respectively.

6. Screen decomposition algorithms

In this section, we present P-way image-space decomposition algorithms according to the taxonomy given in Section 4. Here P denotes the number of processors. Example 16-way decompositions for all decomposition schemes discussed here are displayed in Figure 6.

6.1. 1D decomposition algorithms

The 1D decomposition schemes use 1D arrays with the exact model. Without loss of generality, only horizontal decomposition is discussed here. Note that the horizontal decomposition preserves the intra-scanline coherency which is a valuable asset for the DVR algorithms utilizing such coherency. The load-balancing problem in the decomposition of 1D domains can be modeled as the *chains-on-chains* partitioning (CCP) problem [2]. In the CCP problem, we are given a chain of work pieces called modules and wish to partition the chain into P subchains, each subchain consisting of consecutive modules. The subchain with the maximum load is called the *bottleneck subchain* and the load of this subchain is called the *bottleneck value* of the partition. The objective is to minimize the bottleneck value. In 1D image-space decomposition, the ith module corresponds to the ith scanline and the load W_{ij} of a subchain $[i, j]$ corresponds to the bb-count of the respective horizontal stripe. W_{ij} is computed in $O(1)$ time by using 1D arrays YS and YE according to Eq. 2.

6.1.1. Heuristic horizontal decomposition (HHD). The screen is decomposed into P horizontal stripes recursively. At each recursive-bisection step, a region $[i, j]$ is divided into two regions $[i, k]$ and $[k + 1, j]$ such that the horizontal division line k minimizes the expression $max\{W_{i,k}, W_{k+1,j}\}$. Although each bisection achieves an optimal division, the sequence of these optimal bisections may lead to poor load balancing. The HHD algorithm runs in $\Theta(M \log P)$ time.

6.1.2. Optimal horizontal decomposition (OHD). The existing algorithms for the optimal solution of the CCP problem can be classified into two categories: dynamic programming and probe-based methods. We use Nicol's probe-based CCP algorithm [21] because of its better speed performance. Nicol's algorithm is based on a *probe* function, which takes a candidate bottleneck value L and determines whether a partition exists with a bottleneck value L', where $L' \leq L$. The probe function loads consecutive processors with consecutive subchains in a greedy manner such that each processor is loaded as much as possible without exceeding L. That is, the probe function finds the largest index i_1 such that $W_{1,i_1} \leq L$ by using the binary search. Similarly, it finds the largest index i_2 such that $W_{1,i_2} \leq W_{1,i_1} + L$. This process continues until either all modules are assigned, or some modules remain after loading P parts. In the former case, we say that a partition with a bottleneck value no more than L exists. In the latter case, we know that no partition exists with a bottleneck value less than or equal to L. It is clear that this greedy approach will find a solution with a bottleneck value no greater than L if there is any. The probe function runs in $O(P \log M)$ time.

Nicol's algorithm exploits the fact that candidate L values all have the form of W_{ij} for $1 \leq i \leq j \leq M$. It efficiently searches for the earliest range W_{ij} for which $L_{opt} = W_{ij}$ by considering each processor in order as a candidate bottleneck processor in an optimal mapping. The algorithm uses the binary search and finds the greatest index i_1 such that the call of probe with W_{1,i_1} returns false. Here, either W_{1,i_1+1} is the bottleneck value of an optimal solution, which means that processor 1 is a bottleneck processor, or W_{1,i_1} is the cost of processor 1 in an optimal solution. We save W_{1,i_1+1} as L_1, and starting from $i_1 + 1$ perform the same operation to find i_2 and L_2. This operation is repeated P times and the minimum of L_i values for $1 \leq i \leq P$ constitutes the optimal bottleneck value. Once the optimal bottleneck value is found, the division points for an optimal decomposition can easily be found by using the greedy approach of the probe function. The complexity of this probe-based CCP algorithm is $O(M + (P \log M)^2)$, where the $\Theta(M)$ cost comes from the initial prefix-sum operation on the YS and YE arrays.

6.2. 2D decomposition algorithms

6.2.1. Rectilinear decomposition (RD). This scheme is a 2D decomposition algorithm using 2D arrays with the exact model. This scheme divides the y-dimension into p horizontal stripes and the x-dimension into q vertical stripes, where $p \times q = P$. We use Nicol's iterative algorithm [21] that utilizes CCP for rectilinear decomposition. As the optimal rectilinear decomposition is known to be NP-complete [9], this algorithm finds a suboptimal solution.

The iterative algorithm is based on finding an optimal partition in one dimension given a fixed partition in the alternate dimension. The next iteration uses the partition just found in the previous dimension as a fixed partition and finds an optimal partition in the other dimension. This operation is repeated, each time fixing the partition in the alternate dimension. The decomposition problem in one dimension

is the adaptation of the CCP problem. It is proven that the iterations converge very fast to a locally optimum solution [21].

The original algorithm has $O(K(n^2 + (pq \log n)^2))$-time complexity for an $n \times n$ square coarse mesh, where K is the number of iterations. Here, the $\Theta(n^2)$ cost comes from collapsing stripes into chains. In this work, we use our exact model proposed in Section 5.2.2 to query the bb-count of a rectangular region in $O(1)$ time. This scheme avoids the need for collapsing at each iteration, thus reducing the overall complexity to $O(n^2 + K(pq \log n)^2)$. We also note that this asymptotic improvement proposed for image-space decomposition is also valid for the rectilinear decomposition of general 2D workload arrays. In the general case, the $n \times n$ workload array should be converted into a SAT by performing a 2D prefix-sum operation in $\Theta(n^2)$ time at the beginning.

6.2.2. Jagged decomposition (JD). In the jagged decomposition, the screen is partitioned into p horizontal stripes, and each horizontal stripe is independently partitioned into q vertical stripes, where $p \times q = P$.

Heuristic Jagged Decomposition (HJD). This algorithm is a 2D decomposition algorithm using 1D arrays with the exact model. In this scheme, the screen is divided into p horizontal stripes through a recursive-bisection based heuristic as in the HHD scheme using 1D YS and YE arrays. After this 1D decomposition along the y-dimension, the workload distribution in the x-dimension is calculated for each horizontal stripe using two 1D arrays XS and XE. Then, each horizontal stripe is divided independently in parallel into q vertical stripes in the x-dimension using the same recursive-bisection based heuristic. The HJD algorithm runs in $\Theta(M \log p + N \log q)$ time since the vertical partitioning of the horizontal stripes are performed in parallel.

Optimal Jagged Decomposition (OJD). The OJD algorithms presented in this section are based on the optimal semi-generalized block partitioning algorithm proposed by Manne and Sørevik [17]. Their algorithm extends the dynamic-programming based 1D CCP [16] to OJD. They perform a p-way CCP on the rows of the workload array. The cost of a subchain in a p-way rowwise partition is found by applying a q-way CCP on the columns of that stripe and taking the bottleneck value of the columnwise partition as the cost of the subchain. The complexity of their algorithm is $O(pq(m - p)(n - q))$. In this work, we extend Nicol's probe-based CCP algorithm to OJD of general 2D workload arrays. A straightforward extension of Nicol's CCP algorithm leads to an $O((p \log m)^2(mn + (q \log n)^2))$-time OJD algorithm. Fortunately, the run time of our algorithm asymptoticly reduces to $O(mn + (p \log m)^2(q \log n)^2)$ by converting the 2D $m \times n$ workload array into a SAT through an initial 2D prefix-sum operation in $\Theta(mn)$ time as mentioned earlier. The details of the proposed OJD algorithm can be found in [13].

The proposed OJD algorithm is exploited in two distinct image-space decomposition schemes, namely OJD with the IAH model (OJD-I) and OJD with the exact model (OJD-E). Note that the OJD-I scheme will create a suboptimal jagged decomposition for actual primitive distribution because of the IAH model used.

6.2.3. Orthogonal recursive bisection (ORB). We present four algorithms based on the orthogonal recursive bisection paradigm. These algorithms divide a region into two subregions so that the workload of the most heavily loaded subregion is minimized. Then, they recursively bisect the subregions, each time dividing along the longer dimension, until the number of regions is equal to the number of processors. Dividing the longer dimension aims at reducing the total perimeter of the final regions as an attempt to reduce the amount of primitive replication due to the primitives intersecting multiple regions.

ORB with 1D arrays (ORB-1D). This scheme is a 2D decomposition algorithm using 1D arrays with the exact model. In this scheme, the primitive distribution over both dimensions of the screen is needed to divide the screen horizontally or vertically. 1D arrays are used for each dimension of the screen. That is, in addition to the YS and YE arrays for the y-dimension, each processor allocates XS and XE arrays for the x-dimension of the screen.

In this scheme, the initially created workload arrays are updated between the successive steps of the decomposition operations. Therefore, the global-sum and prefix-sum operations are repeated after each update of the workload arrays [1]. Initially, each processor is assigned the full screen as its local image region. Each processor divides its local image region into two regions along the longer dimension. Note that for the group of processors that are assigned to the same image region, the division will be the same. After the division, half of the processors are assigned to one of the regions, and the other half of the processors are assigned to the other region. Then, each processor sends the local bounding boxes intersecting the other region to the neighbor processor assigned to that region. Neighborhood between processors can be defined according to various criteria such as interconnection topology of the architecture, labeling of the processors etc. In this work, the hypercube labeling is selected for the neighborhood definition since it is very simple. After this exchange operation, each processor has all bounding boxes that project onto its new local screen region and the decomposition operation is repeated for the new image region. In order to decompose the new screen region, we need to update the (X/Y)S and (X/Y)E arrays for the new region. We update these arrays incrementally using the bounding boxes exchanged between the processors. Each processor decrements the appropriate positions in its local (X/Y)S and (X/Y)E arrays for each bounding box its sends and increments the appropriate locations in these arrays for each bounding box it receives.

Each recursive-bisection level of the ORB-1D algorithm consists of two phases: subdivision, and bounding-box classification and migration. The overall complexity of the subdivision phase is $\Theta(N \log P)$ for a square screen of resolution $N \times N$. Under average-case conditions, half of the bounding boxes can be assumed to migrate at each recursive-bisection level of the algorithm. Hence, if primitive replication is ignored, the total volume of communication due to the bounding-box migrations will be $(B/2) \log P$ bounding boxes, where B denotes the total number of bounding boxes in the scene. Under perfect load balance conditions, each processor is expected to hold B/P bounding boxes and each processor pair can be assumed to exchange $B/(2P)$ bounding boxes at each level. Hence, under these conditions, to-

tal concurrent volume of communication will be $(B/(2P))\log P$. Note that primitive replication will increase these values.

ORB with IAH model (ORB-I). This scheme is a 2D decomposition algorithm using 2D arrays with the IAH model. ORB-I is equivalent to the mesh-based adaptive hierarchical decomposition scheme (MAHD) proposed by Mueller [20]. The bisection lines for dividing along the longer dimensions are found by using the binary search on the SAT. The ORB-I algorithm runs in $\Theta(n\sqrt{P})$ time for a square coarse mesh of size $n \times n$.

ORB with medians-of-medians (ORBMM). The algorithms presented in this section are 2D decomposition algorithms using 2D arrays with the IAH model. In this scheme, the screen is divided using ORB with the medians-of-medians approach (ORBMM) [23, 27]. The medians-of-medians scheme is used to decrease the load imbalance at each recursive decomposition step by relaxing the straight line restriction in the division. We apply ORBMM on a cartesian mesh (ORBMM-M) and a quadtree (ORBMM-Q).

In ORBMM-Q, we generate a quadtree from the mesh superimposed on the screen to decrease the errors due to the IAH model. Each leaf node of the tree is referred to here as a *quadnode*. The quadtree is generated in such a way that each quadnode has approximately the same workload. The screen is decomposed at the quadnode boundaries. In this work, we use a bottom-up approach to generate a quadtree, whereas a top-down approach was implemented in [11]. That work uses a 2D segment tree [24] to generate the quadtree. The 2D segment tree data structure has a recursive structure. In the top-down approach, the root of the tree covers the full screen and at each level a node is divided into four quadrants forming its children. This division is repeated until the size of a leaf node is equal to one mesh cell. Then, the segment tree is traversed to generate the quadnodes. In the bottom-up approach, we join the mesh cells to form the quadnodes. The mesh cells may be considered as the leaf nodes of the segment tree. In this scheme, we do not explicitly create a segment tree. Instead, we consider a virtual segment tree superimposed on the coarse mesh. This virtual segment tree is traversed in a bottom-up fashion and the quadnodes are inserted into a linked-list structure. In this traversal, if the cost of a node is under a specified threshold value, but one of the siblings exceeds the threshold value, we add this node to the quadnode list. In addition, if four sibling nodes do not individually exceed the threshold value, but their sum exceeds, we add four of them to the quadnode list. The threshold value is selected empirically as a certain percentage of the total number of primitives. Larger threshold values cause poor load balance, whereas smaller values result in more irregular partitions and larger decomposition time.

In ORBMM-M, the cells of the coarse mesh are inserted into a linked-list structure in a similar way to that of ORBMM-Q. This scheme may also be considered as a special case of the quadtree scheme, where the threshold value is zero and each leaf node is inserted into the linked-list structure.

At each recursive-bisection step of ORBMM, the mid-points of the quadnodes are used to find the median line. The problem with this scheme is how to assign the

quadnodes which straddle the median-line. Taking the centers of the quadnodes and assigning them according to their centers may cause load imbalance. The medians-of-medians (MM) scheme [23, 27] is used to alleviate this problem. The idea in the MM scheme is once the median line is determined, the border quadnodes that straddle the median-line are identified and repartitioned. In this phase, we sort the border quadnodes along the bisection direction and assign the nodes in this order to the one side so that the load of the maximally loaded side is minimized. Both ORBMM-Q and ORBMM-M run in $O(n^2\sqrt{P})$ time for a square coarse mesh of size $n \times n$.

6.2.4. Hilbert-curve based decomposition (HCD). This scheme is a 2D decomposition algorithm using 2D arrays with the IAH model. In the HCD scheme, the 2D coarse mesh is traversed according to the Hilbert curve to map the 2D coarse mesh to a 1D chain of mesh cells. That is, the mesh cells are assigned to the processors such that each processor gets the cells that are consecutive in this traversal. The load-balancing problem in this decomposition scheme reduces to the CCP problem. The advantage of the Hilbert curve over the other space-filling curves [19, 23] is that it traverses the 2D mesh along neighbor cells without any jumps. Therefore, we may expect that the total perimeter of the resulting regions will be less compared to the regions to be obtained by other curves. An example for traversing a 2D mesh with the Hilbert curve is shown in Figure 4(a). The numbers inside the cells represent the order the mesh cells are traversed.

Our approach to traverse the mesh is based on the costzones scheme proposed by Singh et al. [27]. Their approach traverses the 2D segment tree already superimposed on the space. Here, we superimpose a virtual 2D segment tree on the screen. The key idea in this approach is to traverse the child nodes in a predetermined order such that the traversal of the leaf nodes forms a space-filling curve. As the children of a node are the quadrants of the region represented by that node, we have four possible starting points and two possible directions (clockwise or counterclockwise) for each starting point. An appropriate choice of four out of the eight orderings is needed. The ordering of the children of a cell C depends on the order-

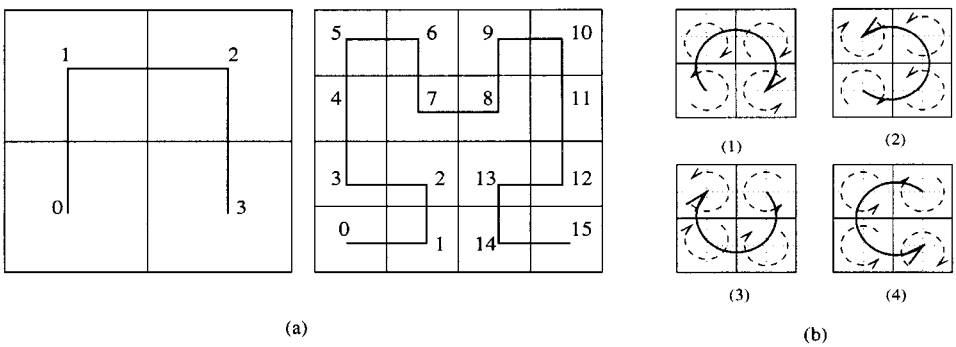


Figure 4. (a) Traversing a 2D mesh with the Hilbert curve and mapping of the mesh-cell locations into 1D array indices. (b) Child ordering of the costzones scheme.

ing of C's parent's children and the position of C in this ordering (Fig. 4(b)). The cells are assigned to the processors during the traversal of the virtual 2D segment tree.

Mapping the 2D coarse mesh to a 1D chain of mesh cells takes $\Theta(mn)$ time and the recursive-bisection based heuristic used for load-balanced partitioning takes $\Theta(mn \log P)$ time.

6.2.5. Graph-partitioning based decomposition (GPD). This scheme is a 2D decomposition algorithm using 2D arrays with the IAH model. This algorithm models the image-space decomposition problem as a graph-partitioning (GP) problem [12]. Each cell in the coarse mesh is assumed to be connected to its north, south, west and east neighbors. The vertices of the graph are the mesh cells and the conceptual connections between the neighbor mesh cells form the edges of the graph. The weight of a cell represents the bb-count of the cell. The weight of an edge between two neighbor cells represents the number of bounding boxes intersecting both cells. The objective in GP is to minimize the cutsize among the parts while maintaining the balance among the part sizes. Here, the cutsize refers to the sum of the weights of the cut edges, where an edge is said to be cut if its pair of vertices belong to two different parts. The size of a part refers to the sum of the weights of the vertices in that part. In our case, maintaining the balance among the part sizes corresponds to maintaining the computational load balance. Minimizing the cutsize corresponds to minimizing the amount of primitive replication due to the shared primitives. The state-of-the-art GP tool MeTiS [10] is used in this work. The GP approach decomposes the screen in the most general way. Unlike the previous decomposition algorithms, noncontiguous sets of cells may be assigned to a processor.

In the GPD scheme, each processor tallies the bounding boxes of its local primitives to the mesh cells and updates the weights of the corresponding cells and edges. The cell weights are updated using the IAH model. In order to decrease the errors caused by the primitives shared among more than two cells, we adopt the following scheme for edge weighting. First, we classify the shared bounding boxes into three categories: vertical, horizontal, and general. The vertical bounding boxes are the ones that intersect multiple cells in only a single column of the coarse mesh. The horizontal bounding boxes intersect multiple cells in only a single row. The general bounding boxes intersect multiple (at least 4) cells in different rows and columns. The weight of an edge between two neighbor cells in a row (column) is incremented by a value proportional to the number of horizontal (vertical) bounding boxes intersecting those two cells. The weight of an edge between two neighbor cells is incremented by each general bounding box intersecting those two cells with a value inversely proportional to the total number of cells the bounding box intersects.

The average-case running time of recursive-bisection based pMeTiS can be assumed to be $O(e \log P)$, where e denotes the number of edges in the given graph. Since the graph constructed in the GPD scheme contains $4mn - 2(m + n)$ edges, the GPD algorithm runs in $O(mn \log P)$. However, the hidden constants in the complexity are substantially larger than those of the other decomposition algorithms.

7. Primitive redistribution

After the decomposition of the screen, each processor needs all the primitives intersecting the region it is assigned in order to perform the local rendering calculations. Thus, the local primitives in each processor should be redistributed according to the region-to-processor assignment. Each processor classifies its local primitives according to the regions they intersect. According to the classification, each primitive is stored in the respective send buffer of that region. If a primitive overlaps multiple regions, the primitive is stored in the send buffers of the respective regions. These buffers are exchanged to complete the redistribution of the primitives. In this work, we propose several algorithms for classifying primitives in the redistribution phase.

7.1. 2D mesh-based primitive classification

This is the most general classification scheme applicable to all decomposition schemes. In this scheme, each mesh cell constituting a pixel block is marked with the index of the processor whose screen region covers this particular cell. Then, each bounding box is tallied to the mesh cells, and the respective primitive is stored into the send buffers of the processors according to the marks of the cells the bounding box covers. This scheme is computationally expensive since it involves tallying the bounding boxes, and its complexity per primitive is proportional to the area of the bounding box of the primitive. This scheme is adopted in the HCD, ORBMM-Q, ORBMM-M and GPD algorithms, since these algorithms produce non-rectangular regions.

7.2. Rectangle-intersection based primitive classification

The expensive tallying operation needed in the mesh-based classification can be avoided for the decomposition algorithms that generate rectangular regions. The intersection of a primitive with a rectangular region can easily be tested at a very low cost by testing the intersection of both the x and y extents of the primitive with the x and y extents of the region, respectively. This classification scheme is adopted in the ORB-1D and ORB-I decomposition algorithms that generate rectangular regions in a hierarchical manner based on recursive bisection. The hierarchical nature of these two decomposition schemes is exploited to reduce the number of intersection tests for a bounding-box b to $O(R_b \log P)$, where R_b denotes number of regions intersecting b.

7.3. Inverse-mapping based primitive classification

We propose more efficient algorithms for the 1D horizontal, 2D rectilinear and 2D jagged decomposition schemes by exploiting the regularity of the resulting decompositions. In these schemes, it is possible to give numbers to the resulting regions in row-major (column-major) order if the screen is divided completely in the

y-dimension (x-dimension). For example, in the row-major ordering of a $p \times q$ rectilinear decomposition, the region at the rth horizontal stripe and cth vertical stripe is labeled as $q(r-1)+c$ for $r = 1, \ldots, p$ and $c = 1, \ldots, q$.

In the 1D decomposition schemes, we use an inverse-mapping array IMY of size m. This array represents the assignment of the scanlines to the processors. That is, $\text{IMY}[i] = k$ for each scanline i in the kth horizontal stripe. Then, a primitive with y-extent $(y_{\min}, y_{\max})$ is inserted into the send buffer of each processor k for $\text{IMY}[y_{\min}] \leq k \leq \text{IMY}[y_{\max}]$. This algorithm is faster than the rectangle-intersection based scheme, because only two simple table lookups are sufficient to classify a primitive.

The rectilinear decomposition scheme requires two inverse-mapping arrays: IMY of size m and IMX of size n. Arrays IMY and IMX represent the assignments of the rows (scanlines) and columns of the screen to the horizontal and vertical stripes of the decomposition. That is, $\text{IMY}[i] = r$ for each row i of the screen in the rth horizontal stripe. Similarly, $\text{IMX}[j] = c$ for each column j of the screen in the cth vertical stripe. A primitive with y-extent $(y_{\min}, y_{\max})$ and x-extent $(x_{\min}, x_{\max})$ is inserted into the send buffer of each processor $k = q(r-1)+c$ for $\text{IMY}[y_{\min}] \leq r \leq \text{IMY}[y_{\max}]$ and $\text{IMX}[x_{\min}] \leq c \leq \text{IMX}[x_{\max}]$. This scheme requires only 4 table lookups for the classification of a primitive.

The jagged decomposition scheme requires $(p+1)$ inverse-mapping arrays. One array IMY of size m is needed for the horizontal striping and one IMX^r array of size n is needed for the vertical striping in the rth horizontal stripe for $r = 1, \ldots, p$. Here, $\text{IMY}[i] = r$ for each row i of the screen in the rth horizontal stripe. $\text{IMX}^r[j] = c$ for each column j in the cth vertical stripe of the rth horizontal stripe. A primitive is inserted into the send buffer of each processor $k = q(r-1)+c$ for $\text{IMY}[y_{\min}] \leq r \leq \text{IMY}[y_{\max}]$ and $\text{IMX}^r[x_{\min}] \leq c \leq \text{IMX}^r[x_{\max}]$. This scheme requires $2h+2$ table lookups for classifying a primitive which intersects h consecutive horizontal stripes.

8. Models for performance comparison

The performance comparison of the image-space decomposition algorithms is conducted on a common framework according to three criteria: load balancing, amount of primitive replication and parallel execution time. Primitive replication is a crucial criterion because of the following reasons. A primitive shared by k processors incurs $k-1$ replications of the primitive after the redistribution. Thus, each replication of a primitive incurs the communication of the primitive during the redistribution phase, and redundant computation and redundant storage in the local rendering phase. Furthermore, as we will discuss in Section 8.2, primitive replication has an adverse effect on the load balancing performance of the decomposition algorithms. The parallel execution time of a decomposition algorithm is also an important criterion since the decomposition operation is a preprocessing for the sake of efficient parallel rendering.

A formal approach for the prediction, analysis and comparison of the load balancing and primitive replication performances of the algorithms requires a prob-

abilistic model. However, such a formal approach necessitates the selection of a proper probabilistic distribution function for representing both the bounding-box and bounding-box size distribution over the screen. In this work, we present a rather informal performance comparison based on establishing appropriate quality measures. The experimental results given in Section 9 confirm the validity of the proposed measures.

8.1. Primitive replication

Among all decomposition algorithms, only the GPD algorithm explicitly tries to minimize the amount of primitive replication. In the other algorithms, the topological properties of the underlying decomposition scheme have a very strong effect on the amount of primitive replication. Here, we establish two topological quality measures: the total perimeter of the regions and the total number of junction points generated with the division lines.

The number of bounding boxes intersecting two neighbor regions is expected to increase with increasing length of the boundary between those two regions. In the 1D decomposition schemes, the total perimeter is equal to $(P-1)N$. In both jagged and rectilinear decomposition schemes, the total perimeter is equal to $(p-1)N+(q-1)M$ for a $p \times q$ processor mesh. For an $N \times N$ square screen, the total perimeter is minimized by choosing p and q such that the resulting $p \times q$ processor mesh is close to a square as much as possible. The total perimeter becomes $2(\sqrt{P}-1)N$ for a $\sqrt{P} \times \sqrt{P}$ square processor mesh in both jagged and rectilinear decomposition schemes. In the ORB and ORBMM decomposition schemes for P being an even power of 2, the total perimeter values are equal to $2(\sqrt{P}-1)N$ and $2(\sqrt{P}-1)N+(P-1)(N/n)$, respectively, if the successive divisions are always performed along the alternate dimensions. Note that the total perimeter of ORB with divisions along the alternate dimensions is equal to that of the jagged and rectilinear schemes. However, in our implementation for the ORB and ORBMM schemes, dividing along the longer dimension is expected to further reduce the total perimeter especially in the decomposition of highly non-uniform workload arrays for large P values. In the HCD scheme, using the Hilbert curve as the space-filling curve is an implicit effort towards reducing the total perimeter since the Hilbert curve avoids jumps during the traversal of the 2D coarse mesh. Nevertheless, the total perimeter in the HCD scheme is expected to be much higher than those of the other 2D decomposition schemes, since HCD generates irregularly shaped regions.

The total perimeter measure mainly accounts for the amount of replication due to the bounding boxes intersecting two neighbor regions under the assumption that both x and y extents of the bounding boxes are much smaller than the x and y extents of the regions respectively. A *junction* point of degree $d > 2$—shared by the boundaries of d regions—is very likely to incur bounding boxes intersecting those d regions thus resulting in $d-1$ replications of the respective primitives. Each junction point can be weighted by its degree minus one so that the sum of the weights of the junction points can be used as a secondary quality measure for the relative performance evaluation of the 2D decomposition schemes. It should be noted here

that the validity of this measure depends on the assumption that the bounding boxes are sufficiently small and the junction points are sufficiently away from each other so that the bounding boxes do not cover multiple junction points. Both jagged and ORB decomposition schemes generate $2(p-1)(q-1)$ junction points each with a degree of 3 for a $p \times q$ processor mesh. Here, we assume that the successive divisions are always performed along the alternate dimensions in the ORB scheme. The number of junction points generated by the rectilinear decomposition scheme is half as much as that of the jagged and ORB schemes. However, each junction point in the rectilinear decomposition scheme has a degree of 4. As seen in Table 2, under the given assumptions and the additional assumption that the bounding-box density remains the same around the junction points, the rectilinear decomposition is expected to incur 25% less primitive replication than both jagged and ORB decomposition schemes due to the bounding boxes intersecting more than two regions.

As seen in Table 2, according to the total perimeter measure, the 2D decomposition schemes are expected to perform better than the 1D schemes such that this performance gap is also expected to increase rapidly with increasing P. Among the 2D decomposition schemes, the jagged, rectilinear and ORB schemes are expected to display comparable performance. However, ORB can be expected to perform slightly better than the other two schemes because of the possibility of further reduction in the total perimeter by performing divisions along the longer dimension. As the jagged and rectilinear decomposition schemes are expected to display similar performance according to the total perimeter measure, the rectilinear decomposition scheme is expected to perform better than the jagged decomposition scheme because of the secondary quality measure.

8.2. *Load balancing*

The load-balancing performance of the decomposition algorithms are evaluated according to the actual load-imbalance values defined as

$$LI = (B'_{\max} - B_{\text{avg}})/B_{\text{avg}} \qquad [5]$$

Table 2. Topological quality measures of the decomposition schemes for primitive replication for a square screen of resolution $N \times N$

Decomposition scheme	Total perimeter	(Degree − 1) weighted junction sum
1D (Horizontal)	$(P-1)N$	0
Rectilinear	$2(\sqrt{P}-1)N$	$3(\sqrt{P}-1)^2$
Jagged	$2(\sqrt{P}-1)N$	$4(\sqrt{P}-1)^2$
ORB	$\leq 2(\sqrt{P}-1)N$	$4(\sqrt{P}-1)^2$
ORBMM	$\leq 2(\sqrt{P}-1)N + (P-1)(N/n)$	$4(\sqrt{P}-1)^2$

P is assumed to be an even power of 2 for 2D decomposition schemes. A square coarse mesh of resolution $n \times n$ is assumed for the 2D-2D algorithms, so that N/n denotes the height and width of a coarse-mesh cell.

for a P-way decomposition of a given screen with B primitives. Here, $B'_{\max}$ denotes the number of primitives (triangles) assigned to the most heavily loaded (bottleneck) processor, and $B_{\text{avg}} = B/P$ denotes the number of primitives to be assigned to each processor under the ideal balance condition without primitive replication. Let

$$r = (B' - B)/B \qquad [6]$$

denote the primitive replication ratio, where B' denotes the total number of primitives in the parallel environment after the primitive redistribution phase so that $B' - B$ corresponds to the total amount of primitive replication. Also let $\varepsilon = (B'_{\max} - B'_{\text{avg}})/B'_{\text{avg}}$ denote the simple load-imbalance ratio, where $B'_{\text{avg}} = B'/P$. Then, the actual load-imbalance ratio of a decomposition with a primitive replication ratio of r and a simple load-imbalance ratio of ε will be

$$LI = \frac{\frac{B'}{P}(1+\varepsilon) - \frac{B}{P}}{\frac{B}{P}} = \frac{\frac{B(1+r)}{P}(1+\varepsilon) - \frac{B}{P}}{\frac{B}{P}} = r + \varepsilon + r\varepsilon. \qquad [7]$$

Note that ε mainly accounts for the simple load-balancing performance of the subdivision algorithm, whereas r accounts for the primitive-replication performance of the underlying decomposition scheme. It is clear from Eq. 7 that the primitive replication ratio constitutes a lower bound on the actual load-imbalance value of a decomposition. So, although all decomposition algorithms try to minimize the bounding-box count of the bottleneck region (i.e. $B'_{\max}$), their load-balancing performances highly depend on the primitive-replication performances of the underlying decomposition schemes. This dependence is expected to be more pronounced in the decomposition algorithms using the IAH model, because the error rate of the IAH model is likely to increase with increasing amount of primitive replication. So, the primitive-replication performance of the underlying decomposition scheme of a partitioning algorithm is a crucial quality measure for the load-balancing performance of the algorithm.

The following factors are important in the performance comparison of different subdivision algorithms using the same decomposition scheme. The exact algorithms finding globally optimal partitions are expected to perform better than the heuristics. For example, OHD and OJD-E are expected to perform better than HHD and HJD, respectively. The subdivision algorithms using the exact workload model are expected to perform better than the ones using the IAH model because of the adverse effect of the primitive replication on the accuracy of the IAH model. For example, ORB-1D and OJD-E are expected to perform better than ORB-I and OJD-I, respectively.

The *solution-space size* is established as a quality measure for comparing the load-balancing performance of the constructive and iterative-improvement based decomposition algorithms. The performances of these algorithms can be expected to increase with increasing size of the solution space. Here, the solution space of a decomposition algorithm refers to the set of distinct feasible screen partitions the best of which is found by the algorithm. For example, the solution-space sizes of

the OHD, RD and OJD algorithms are

$$S_{OHD} = \binom{N-1}{P-1} \qquad S_{RD} = K\binom{n-1}{\sqrt{P}-1} \leq \binom{n-1}{\sqrt{P}-1}^2 \qquad \text{and}$$

$$S_{OJD} = \binom{n-1}{\sqrt{P}-1}^{\sqrt{P}+1}, \qquad [8]$$

respectively. Here, a $\sqrt{P} \times \sqrt{P}$ square processor mesh is assumed for the 2D rectilinear and jagged decomposition algorithms. As seen in Eq. 8, OJD is expected to perform better than RD which is in turn expected to perform better than OHD. In fact, the load-balancing performance of OJD is always better than or equal to that of RD, because the solution space of RD is a subset of that of OJD. Another factor in favor of OJD is the fact that it is an exact algorithm whereas RD is an iterative heuristic.

Two quality measures are established for the decomposition algorithms using recursive-bisection based heuristics: the total degree of freedom in selecting the line segments for bisecting the regions and the imbalance propagation during the successive bisection steps. The load-balancing performance of these algorithms can be expected to increase with increasing degree of freedom in selecting the bisection line segments. The degree-of-freedom values for the HHD, ORB and HJD algorithms are;

$$D_{\text{HHD}} \approx N \log P, D_{\text{ORB}} \approx 3(\sqrt{P}-1)N \qquad \text{and}$$

$$D_{\text{HJD}} \approx 0.5(\sqrt{P}+1)N \log P, \qquad [9]$$

respectively, where P is assumed to be an even power of 2 for the 2D ORB and HJD algorithms. Hence, both of the 2D ORB and HJD algorithms have more degrees of freedom than the 1D HHD algorithm.

Although HJD has more degrees of freedom than ORB, ORB is less susceptible to imbalance propagation than HJD, because ORB uses shorter line segments than HJD during the earlier recursive bisection levels. Consider the imbalance propagation in the HJD and ORB algorithms for uniform jagged and ORB decompositions. Assume that P is an even power of 2 and each bisection step incurs an equal simple load-imbalance ratio of ε_2. Also assume that a division line segment of length N incurs λN shared bounding boxes between the two respective subregions, so that λ effectively denotes the bounding-box density along the division lines. HJD uses division lines of length N at each bisection step during the first half of the recursion levels, and it uses lines of length $N/\sqrt{P} = N/2^{(\log P)/2}$ at each bisection step during the second half of the recursion levels. ORB uses line segments of length $N/2^{\lfloor \ell/2 \rfloor}$ at each bisection step in recursion level $\ell = 1, 2, \ldots, \log P$. For example, for $P = 16$, HJD and ORB use line segments of lengths N, N, $N/4$, $N/4$ and N, $N/2$, $N/2$, $N/4$ in recursion levels $\ell = 1, 2, 3, 4$, respectively. So, the upper bounds on the actual load-imbalance values for the HJD and ORB heuristics are;

$$LI_{HJD} = \big((1+\varepsilon_2)^4 - 1\big) + \lambda N\big(\tfrac{1}{16}(1+\varepsilon_2)^4 + \tfrac{1}{8}(1+\varepsilon_2)^3 + \tfrac{1}{16}(1+\varepsilon_2)^2 + \tfrac{1}{8}(1+\varepsilon_2)\big) \qquad [10]$$

$$LI_{ORB} = \big((1+\varepsilon_2)^4 - 1\big) + \lambda N\big(\tfrac{1}{16}(1+\varepsilon_2)^4 + \tfrac{1}{16}(1+\varepsilon_2)^3 + \tfrac{1}{8}(1+\varepsilon_2)^2 + \tfrac{1}{8}(1+\varepsilon_2)\big). \quad [11]$$

Thus, the load imbalance value of HJD can be expected to be greater than that of ORB by the amount of $\Delta LI \approx LI_{HJD} - LI_{ORB} = \varepsilon_2(1+\varepsilon_2)^2\lambda N/16$. The difference in the load imbalance is expected to increase with increasing P. For example, for $P = 64$, HJD and ORB use line segments of lengths N, N, N, $N/8$, $N/8$, $N/8$ and N, $N/2$, $N/2$, $N/4$, $N/4$, $N/8$ in recursion levels $\ell = 1, 2, 3, 4, 5, 6$, respectively, thus resulting in $\Delta LI \approx \varepsilon_2(1+\varepsilon_2)^2(2+\varepsilon_2)(3+\varepsilon_2)\lambda N/64$.

8.3. Execution time

Table 3 displays the dissection of the parallel execution times of the decomposition algorithms. Since the bounding-box creation time is equal ($\Theta(B/P)$) in all decomposition algorithms, it is not displayed in the table.

As seen in Table 3, the workload array creation time (T_{WL}) can be further dissected into two components: the local workload array fill-in time (T_F) and the prefix-sum operation time (T_{PSUM}). As seen in the table, T_F decreases linearly with increasing number of processors as the fill-in computations are performed in parallel without communication. As seen in Table 3, both T_{PSUM} and global-sum operation time (T_{GS}) constitute unscalable components in all algorithms. This unscalability is more severe in the 2D decomposition algorithms using 2D arrays. Imposing a coarse mesh on the screen is an effort towards maintaining these unscalable components within reasonable limits in these algorithms. T_{WL} and T_{GS} of the decomposition algorithms using 2D arrays are significantly more than those of the algorithms using 1D arrays. Among the algorithms using 2D arrays, the ones utilizing the exact model incur six times T_{PSUM} and four times T_{GS} than the ones utilizing the IAH model. However, during the workload array fill-in operations, the tallying time for a bounding-box b is proportional to its area a_b in the IAH model, whereas it is proportional to half of its perimeter $h_b + h_w$ in the exact model. Thus, T_F in the IAH model is more sensitive to the increase in the mesh resolution than T_F in the exact model.

As seen in Table 3, the subdivision times (T_S) of all decomposition algorithms increase with increasing number of processors since more divisions are performed over the same workload array. Note that the subdivision phases of only the 2D-1D based HCD and ORB-1D algorithms are implemented in parallel through exploiting their parallel nature. As seen in the table, the OHD and OJD algorithms, which utilize the efficient probe-based CCP schemes, find globally optimal horizontal and jagged decompositions with comparable run-time complexities to those of the HHD and HJD algorithms which utilize recursive-bisection based heuristics. The asymptotic efficiency of the OJD algorithm is mainly because of the proposed SAT scheme which enables the probe-based algorithm to query the workload of a rectangular region in $O(1)$ time. Similarly, the SAT scheme brings an asymptotic efficiency to the RD algorithm. The comparison of the run-time complexities of the ORB-I and

Table 3. The dissection of the parallel execution times of the image-space decomposition algorithms for a screen of resolution $N \times N$ and a coarse mesh of resolution $n \times n$

Algorithm	Classification	T_{WL}	T_{GS}	T_S	T_{PC}
HHD				$O(N \log P)$	$2\frac{B}{P}t_{lup}$
OHD	1D-1D-exact	$2\frac{B}{P}t_{inc} + 2Nt_{psum}$	$2\frac{P-1}{P}Nt_{com}$	$O(P^2 \log^2 N)$	$2\frac{B}{P}t_{lup}$
HJD		$\frac{B}{P}(4+H_b)t_{inc} + 2(\sqrt{P}+1)Nt_{psum}$	$2(\sqrt{P}+1) \times \frac{P-1}{P}Nt_{com}$	$O(N \log P)$	$(2H_b+2)\frac{B}{P}t_{lup}$
ORB-1D	2D-1D-exact	$4\frac{B}{P}t_{inc} + 2N \log Pt_{psum}$	$\leq 2N \log Pt_{com}$	$\geq \frac{B}{2P} \log Pt_{com} + O(N \log P)$	$R_b\frac{B}{P} \log Pt_{comp}$
OJD-I				$O(P^2 \log^4 n)$	$(2H_b+2)\frac{B}{P}t_{lup}$
ORB-I				$O(n\sqrt{P})$	$R_b\frac{B}{P} \log Pt_{comp}$
ORBMM-Q				$O(n^2\sqrt{P})$	$\frac{B}{P}a_bt_{lup}$
ORBMM-M	2D-2D-IAH	$\frac{B}{P}a_bt_{inc} + n^2t_{psum}$	$\frac{P-1}{P}n^2t_{com}$	$O(n^2\sqrt{P})$	$\frac{B}{P}a_bt_{lup}$
HCD				$O(n^2 \log P)$	$\frac{B}{P}a_bt_{lup}$
GPD				$O(n^2 \log P)$	$\frac{B}{P}a_bt_{lup}$
OJD-E				$O(P^2 \log^4 n)$	$(2H_b+2)\frac{B}{P}t_{lup}$
RD	2D-2D-exact	$\frac{B}{P}(2+h_b+w_b)t_{inc} + 6n^2t_{psum}$	$4\frac{P-1}{P}n^2t_{com}$	$O(KP^2 \log^2 n)$	$4\frac{B}{P}t_{lup}$

B denotes the total number of bounding boxes. P, which denotes the number of processors, is assumed to be an even power of 2. T_{WL}, T_{GS}, T_S and T_{PC} denote the local workload array creation time, global-sum operation time on local workload arrays, subdivision time and primitive classification time, respectively. t_{inc} and t_{psum} denote the time taken for incrementing a workload array entry and an addition operation during the prefix-sum, respectively. t_{com} denotes the per-word transmission time. t_{comp} and t_{lup} denote the time taken for testing the intersection of two rectangles and a table lookup operation, respectively. h_b, w_b and a_b denotes the average height, width and area of a bounding-box in terms of number of mesh cells. H_b and R_b denote the average number of horizontal stripes and regions intersecting a bounding box, respectively. K denotes the number of iterations required for convergence in RD.

ORBMM algorithms in Table 3 shows that the use of the medians-of-medians approaches in the ORBMM algorithms asymptotically increases the complexity. It should be noted here that the hidden constants in the asymptotic complexity of the graph-partitioning tool MeTiS used in the GPD scheme are quite large.

As seen in Table 3, the primitive classification time (T_{PC}) of each decomposition algorithm decreases with increasing number of processors due to the parallel nature of the primitive classification operations. The algorithms using the inverse-mapping based classification scheme result in much less T_{PC} than the other algorithms.

Among the algorithms using the inverse-mapping based classification scheme, the regularity of the underlying decomposition scheme dictates the complexity of T_{PC}. That is, the 1D horizontal decomposition scheme incurs less T_{PC} than the 2D rectilinear decomposition scheme which in turn incurs less T_{PC} than the 2D jagged decomposition scheme.

9. Experimental results

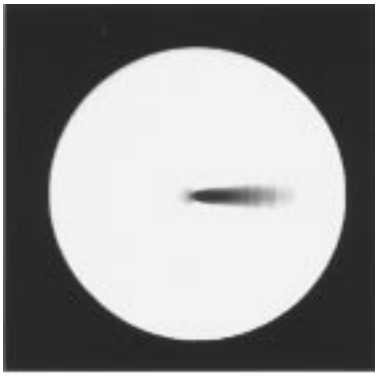
The image-space decomposition and parallel DVR algorithms presented in this work were implemented on a Parsytec CC system, which is a message-passing distributed-memory architecture. The Parsytec CC system contains 16 nodes each of which is equipped with a 133 MHz PowerPC 604 processor and 64 MB memory. The interconnection network consists of sparsely connected four 8×8 crossbar switching boards such that each switching board connects 4 processors. The network can sustain 20 MB/s point-to-point communication bandwidth [14]. The algorithms were implemented using the C language and the native message passing library Embedded Parix (EPX) [22].

Table 4 summarizes the properties of the volumetric datasets used in the experiments. Figure 5 shows the rendered images of these datasets. These datasets are obtained from NASA-Ames Research Center, and they are commonly used by researchers in the volume rendering field. All datasets are originally curvilinear in structure, and they represent the results of CFD simulations. The raw datasets consist of hexahedral cells. They are converted into unstructured tetrahedral data format by dividing each hexahedral cell into 5 tetrahedral cells [8, 26]. Each one of the three datasets is visualized from six different viewing directions at the screen resolution of 512×512. The six viewing directions are selected such that different views of the datasets are rendered as much as possible. The average sequential rendering time of each dataset is displayed in the last column of Table 4. In all figures given in this section, each parallel performance value represents the averages of the results of the eighteen distinct visualization instances for a fixed number of processors (i.e. 3 datasets each of which visualized from 6 viewing directions for a fixed P).

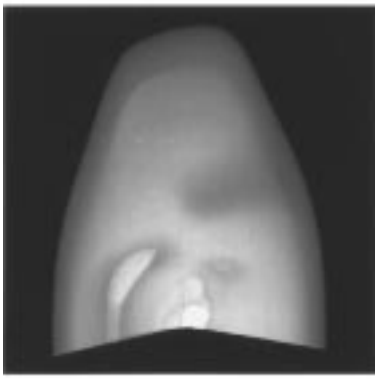
The performances of the decomposition algorithms are experimented on a common framework according to the three criteria discussed in Section 8: percent load imbalance, percent primitive replication and parallel execution time. The percent load-imbalance and percent primitive-replication values are computed as $100LI$ and $100r$, where LI and r are computed according to Eqs. 5 and 6, respectively. The

Table 4. Volumetric datasets used in the experiments

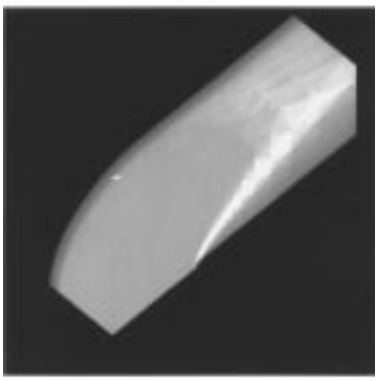
Dataset	# Vertices	# Cells	# Triangles	Seq. rendering time (sec)
Blunt fin	40,960	187,395	381,548	59.6
Oxygen post	109,744	513,375	1,040,588	81.6
Delta wing	211,680	1,005,675	2,032,084	103.6



(a)



(b)



(c)

Figure 5. (a) Oxygen liquid post image, (b) delta wing image, and (c) blunt fin image.

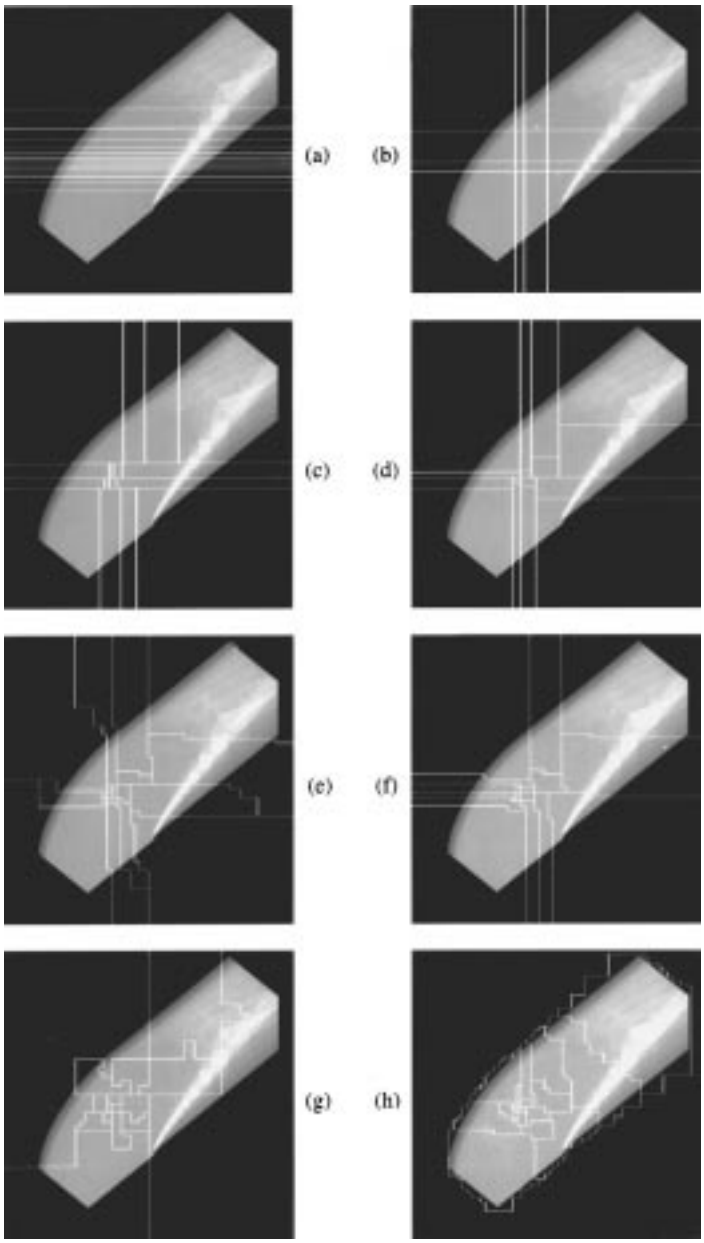


Figure 6. 16-way decompositions produced by the algorithms: (a) HHD and OHD, (b) RD, (c) HJD, OJD-I and OJD-E, (d) ORB-I and ORB-1D, (e) ORBMM-Q, (f) ORBMM-M, (g) HCD, and (h) GPD.

performances of the image-space decomposition algorithms for the sort-first parallelization of Challinger's DVR algorithm are displayed as speedup values. The measured execution times of the decomposition algorithms and the measured speedup values for the parallel renderings are given for 2, 4, 8 and 16 processors of our Parsytec CC system. The percent load-imbalance and percent primitive-replication values are computed and displayed for 2, 4, 8 and 16 processors. These values are also computed and displayed for 32, 64 and 128 processors through simulation to predict and analyze the performance of the decomposition algorithms on large scale parallelization.

The highest screen resolution of 512×512 is used in the algorithms using 1D workload arrays. As mentioned earlier, for the algorithms using 2D workload arrays, a 2D coarse mesh is superimposed on the screen to make the decomposition affordable both in terms of space and execution time. So, the variation of the load-balancing and parallel run-time performances of the 2D decomposition algorithms with varying coarse mesh size is experimented for the mesh resolutions of 32×32, 64×64, 128×128, 256×256 and 512×512. The coarse-mesh size of 512×512 is used for displaying the variation of the performances of all algorithms with varying the number of processors.

Bar charts are used in all of the following figures for displaying the experimental results. In these figures, the vertical scales are selected such that the performance differences among the algorithms can be seen clearly. In some figures, some bars exceed the scale of the graphs because of the small scales selected for the respective figures. In all figures, the vertical dashed lines denote the division lines for the classification of the algorithms according to the taxonomy given in Table 1.

9.1. Primitive replication

Figure 7 displays the percent primitive replication as the number of processors varies. The amount of primitive replication increases with increasing number of processors because of the increase in the total perimeter and number of junction points.

As seen in Figure 7, the 1D decomposition schemes (HHD, OHD) and the HCD scheme perform substantially worse than the other schemes, as expected. Figure 7 shows that the amount of primitive replication increases linearly with increasing number of processors in the 1D schemes. This experimental finding confirms the validity of the total perimeter measure discussed in Section 8, since the total perimeter is equal to $(P-1)N$ in the 1D schemes. Figure 7 also confirms the expectation that the performance gap between the 1D and 2D decompositions increases rapidly in favor of the 2D schemes.

As seen in Figure 7, among the 2D decomposition algorithms, the algorithms utilizing the exact workload model perform better than the algorithms utilizing the IAH model. This experimental finding is attributed to the following reason. All decomposition algorithms try to minimize the load of the most heavily loaded processor. This common nature of the algorithms also corresponds to an effort towards reducing the amount of primitive replication within the limitations of the underly-

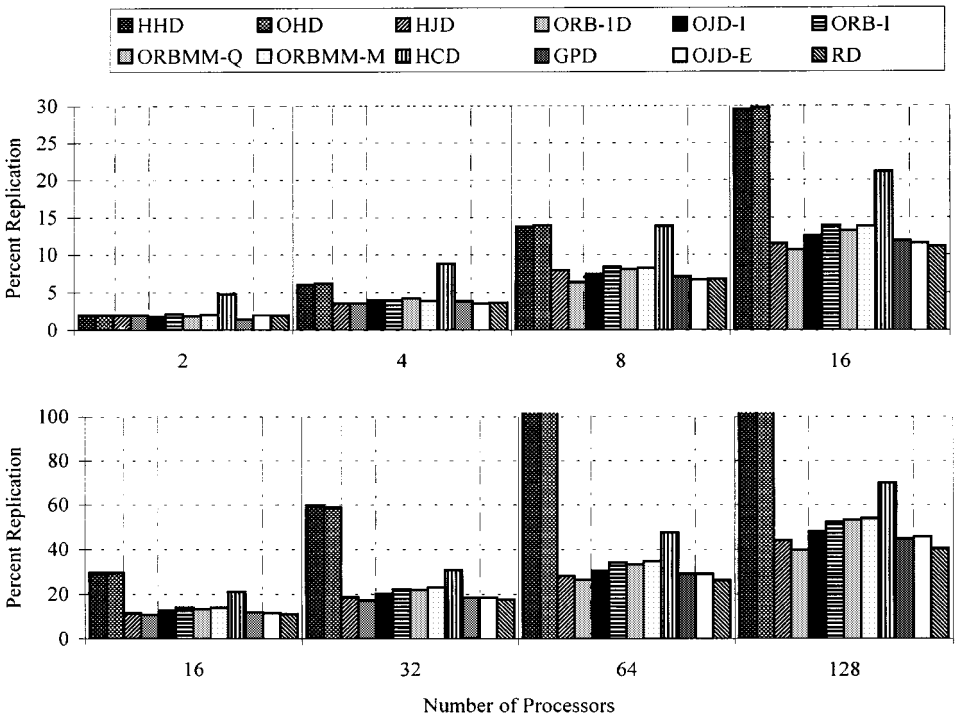


Figure 7. Percent primitive replication on different number of processors.

ing decomposition scheme. Thus, in the algorithms utilizing the IAH model, the errors in the IAH model degrades the primitive replication performance as well as the load-balancing performance of these algorithms.

As seen in Figure 7, among the 2D decomposition algorithms utilizing the exact workload model, ORB-1D performs better than RD which in turn performs better than both HJD and OJD-E. The superior performance of ORD-1D is because of performing divisions along the longer dimension thus leading to further reduction in the total perimeter. Recall that the total perimeter values of the rectilinear and jagged decomposition schemes are the same. Hence, the better performance of RD than the jagged decomposition schemes HJD and OJD-E verifies the validity of the junction-point quality measure discussed in Section 8.

As seen in Figure 7, among the 2D decomposition algorithms utilizing the IAH model, the GPD algorithm displays the best performance since it explicitly tries to minimize the total amount of primitive replication. In fact, the performance of the GPD algorithm approaches to the performance of the 2D decomposition algorithms utilizing the exact workload model. As seen in the figure, HCD performs substantially worse than all other 2D decomposition algorithms, as expected.

9.2. *Load balancing*

Figure 8 illustrates the effect of the coarse-mesh resolution on the load-balancing performance of the 2D decomposition algorithms utilizing 2D workload arrays on 16 processors. The load-balancing performances of all decomposition algorithms are expected to increase with increasing mesh resolution because of the increase in the size of the search space. However, for a fixed number of processors, increasing the mesh resolution is likely to increase the number of primitives intersecting multiple regions thus increasing the error rate of the IAH model. Thus, beyond a certain mesh resolution, the errors due to the IAH model may consume the gain obtained by the increase in the size of the search space. As seen in Figure 8, among the 2D decomposition algorithms using 2D workload arrays with the IAH model, ORBMM-Q, ORBMM-M, HCD and GPD sometimes achieve their best performance values at mesh resolutions less than 512×512, whereas OJD-I and ORB-I always achieve their best results at 512×512. The common characteristic of the former four algorithms is that they all produce non-rectangular regions. As discussed in Section 8.1, division by non-rectangular regions has the potential of increasing the amount of primitive replication because of the increase in the total perimeter. Thus, the experimental results show that errors due to the inverse area heuristic have more adverse affect on the load-balancing performance of the decomposition algorithms that generate non-rectangular regions.

Figure 9 displays the load-balancing performance of the algorithms as the number of processors varies. As expected, load imbalance increases with increasing number of processors. As seen in Figure 9, OJD-E achieves the best load-balancing performance, whereas ORB-1D and HJD display the second and third best performance,

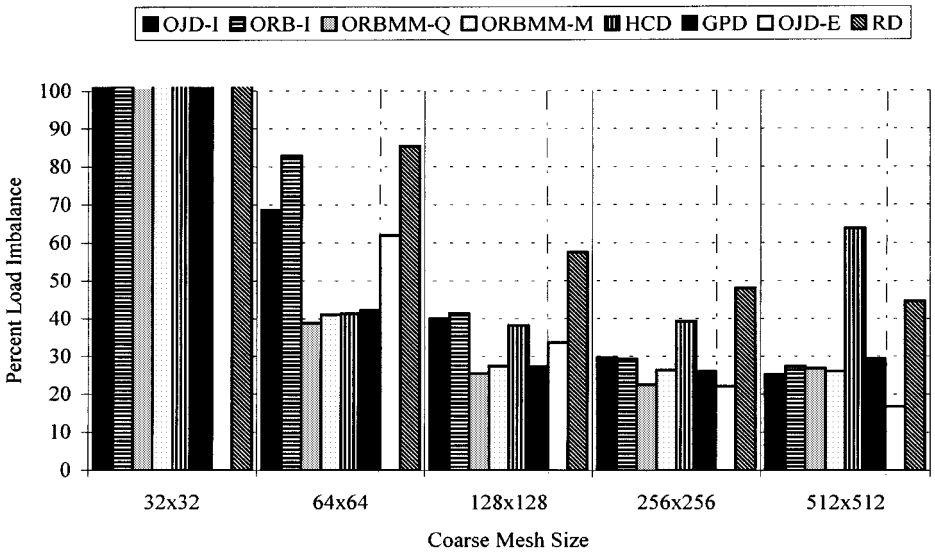


Figure 8. Effect of coarse-mesh resolution on the load-balancing performance of the 2D decomposition algorithms using 2D workload arrays on 16 processors.

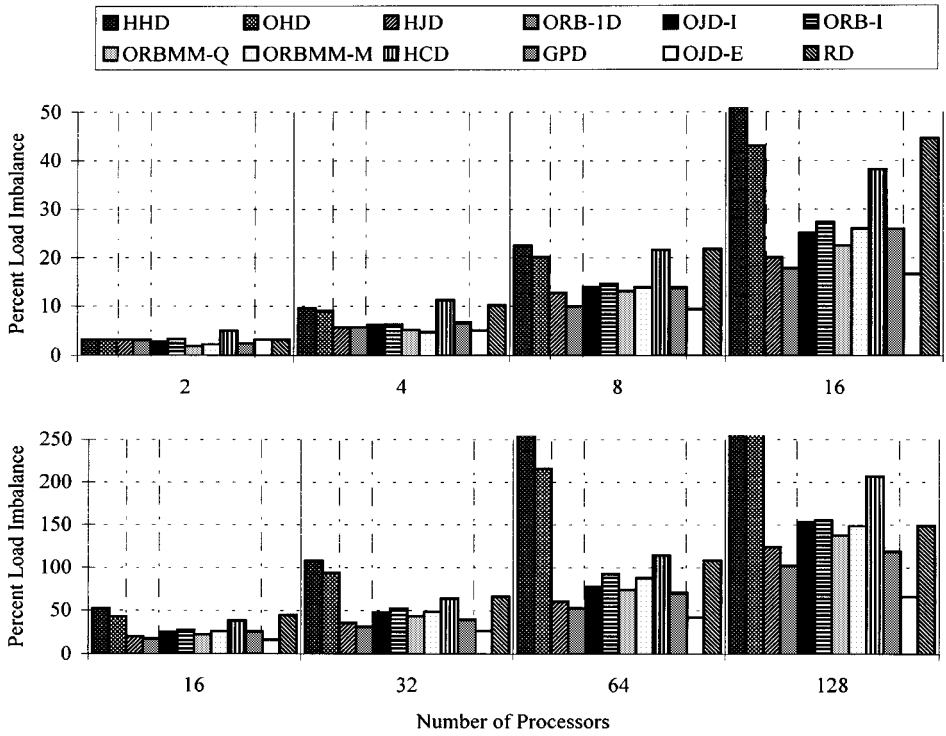


Figure 9. Load-balancing performance of the decomposition algorithms on different number of processors.

respectively. The common characteristics of these three algorithms are that they achieve 2D decomposition, utilize the exact workload model, and produce rectangular regions with almost the same total perimeter values. The superior performance of OJD-E is because of the fact that it is an exact algorithm whereas ORB-1D and HJD are recursive-bisection based heuristics. As seen in Figure 9, the difference in the load-balancing performance among these three algorithms increases with increasing number of processors in favor of OJD-E because of the increase in the amount of imbalance propagation in ORB-1D and HJD with increasing number of recursion levels. The better load-balancing performance of ORB-1D compared to that of HJD is because of the following two reasons. First, as discussed in Section 9.1, ORB incurs less primitive replication than HJD thus resulting in less actual load imbalance. Second, as mentioned in Section 8.2, ORB-1D is less susceptible to imbalance propagation than HJD. Although RD has the same characteristics as these three algorithms, it performs substantially worse than these three algorithms. There are two main reasons for this finding. First, as seen in Eq. 8, rectilinear splits in both dimensions of the screen restrict the solution space in the decomposition. Second, the iterative algorithm may converge to a poor local optimum. Nicol [21] states that starting with many randomly chosen initial partitions and then taking the one giving the best result increases the performance of RD.

As seen in Figure 9, the 1D decomposition algorithms HHD and OHD display very poor load-balancing performance due to following two reasons. First, both HHD and OHD have substantially larger total perimeter value than the 2D decomposition algorithms, thus resulting in the increase of the total primitive replication ratio r in Eq. 7. Second, as mentioned in Section 8.2, HHD and OHD have substantially less degrees of freedom and solution-space size, respectively, than the 2D decomposition algorithms. Among these two, OHD performs better than HHD as expected since OHD finds globally optimum horizontal decompositions.

As seen in Figure 9, the 2D decomposition algorithms which utilize the IAH model display in-between load-balancing performance. Both ORBMM algorithms perform better than the ORB-I algorithm thus showing the merits of the medians-of-medians approach in the ORB-based schemes. The medians-of-medians approach decreases the load imbalance at each recursive bisection step, thus reducing the amount of imbalance propagation in the ORBMM schemes. For small P values, both ORBMM schemes perform better than all the other algorithms utilizing the IAH model. Among the two ORBMM schemes, ORBMM-M performs better than ORBMM-Q for $P = 4$. However, the performance of the quadtree approach becomes better as the number of processors increases. This is because of the fact that the amount of errors due to the inverse area heuristic is less with larger quadnodes. As the number of processors increases, the performance of GPD becomes the best among the ones utilizing the IAH model since it incurs the least amount of primitive replication. The Hilbert-curve based decomposition is found to be not suitable for image-space decomposition, since the HCD scheme displays the worst load-balancing performance among the 2D decomposition schemes.

9.3. Execution time

Figure 10 presents the execution times of the decomposition algorithms with varying the number of processors at the mesh resolution of 512×512. The execution time of each algorithm decreases as the number of processors increases because of the parallel nature of the decomposition algorithms. Figure 10 also illustrates the dissection of the parallel execution times of the decomposition algorithms into five components: bounding-box creation time (T_{BB}), local workload-array creation time (T_{WL}), global-sum operation time on local workload arrays (T_{GS}), subdivision time (T_S) and primitive redistribution time (T_R).

As seen in Figure 10, both T_{BB} and T_{WL} decrease with increasing number of processors as the respective computations are performed in parallel without communication. As seen in the figure, T_{BB} is the dominating component in the overall parallel decomposition time on $P = 4$, and its relative importance decreases with increasing P. In the 2D decomposition algorithms using 2D workload arrays, although the algorithms using the exact model fill four 2D arrays and the ones using the IAH model fill only one 2D array, the former type of algorithms take less workload creation time (T_{WL}) than the latter type of algorithms. This is expected as shown in Table 3, because the tallying time per bounding box is proportional to

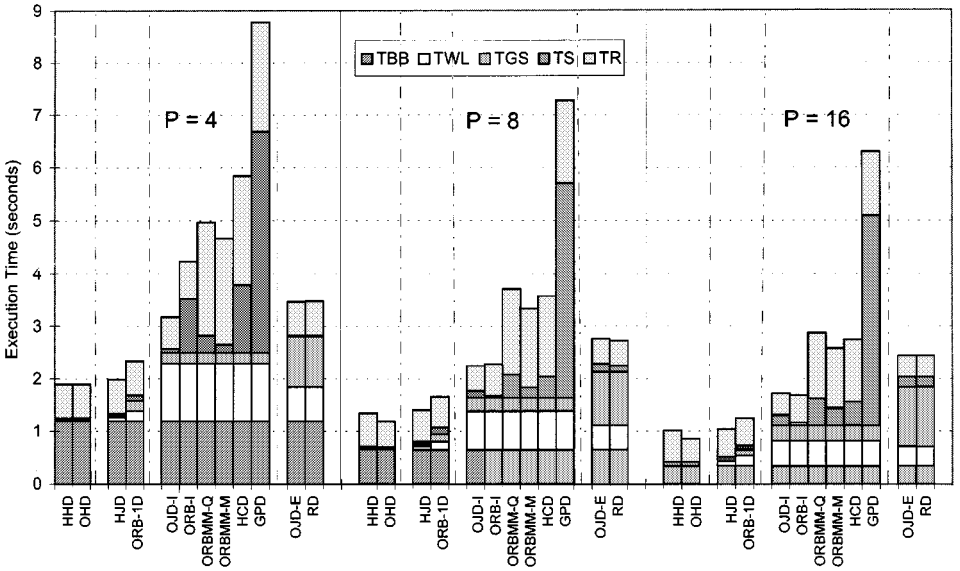


Figure 10. Execution times of the decomposition algorithms on different number of processors at mesh resolution of 512×512. T_{BB}, T_{WL}, T_{GS}, T_S and T_R denote the bounding-box creation, local workload-array creation, global-sum operation, subdivision and primitive redistribution times, respectively.

the area of the bounding box in the IAH model, whereas it is proportional to the perimeter of the bounding box in the exact model. As seen in Figure 10, the global-sum time (T_{GS}) increases slightly with increasing number of processors as expected (see Table 3). In the 2D decomposition algorithms OJD-E and RD utilizing the exact workload model, T_{GS} tends to become the dominant component in the total decomposition time with increasing number of processors, as expected. As seen in Figure 10, the subdivision time (T_S) of each decomposition algorithm increases with increasing number of processors as expected. The redistribution time (T_R) decreases with increasing number of processors as expected since smaller number of primitives are classified at each processor.

The total execution times of the 2D decomposition algorithms with varying coarse-mesh resolution are given in Figure 11. As seen in Figure 11, the execution times increase with increasing mesh resolution as expected. T_{WL}, T_{GS} and T_S increase since the sizes of the local workload arrays increase with increasing mesh resolution. Furthermore, T_R increases in the algorithms that use the mesh-based classification scheme for redistribution, because this classification scheme involves tallying the primitives onto the coarse mesh. As seen in the figure, the largest increases in the execution times occur between the mesh resolutions of 256×256 and 512×512, because the screen-space bounding boxes of the primitives in our datasets are small. Most of them intersect only a few mesh cells (typically one or two cells) at small mesh resolutions. However, at the highest mesh resolution of 512×512, most of the primitives intersect multiple cells.

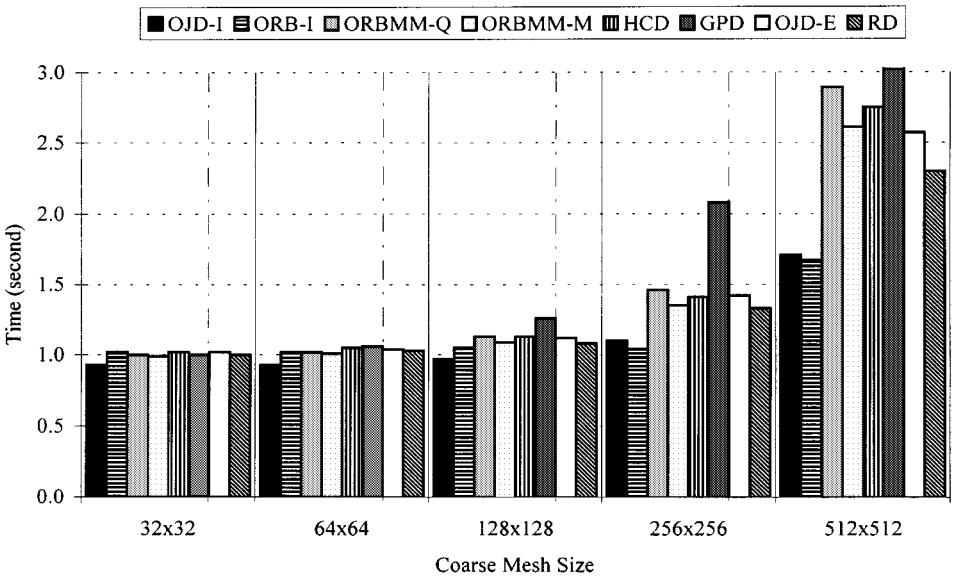


Figure 11. Execution times of the 2D decomposition algorithms using 2D arrays with varying the coarse-mesh resolution on 16 processors.

9.4. Parallel rendering performance

Figures 12–14 display the speedup values obtained in the sort-first parallelization of Challinger's DVR algorithm on the Parsytec CC system. Figure 12 shows the speedup values for the parallel rendering phase when only the number of triangles is used to approximate the workload in a region. In this case, the maximum speedup obtained is 5.9 on 16 processors. Figure 13 illustrates the speedup values for the rendering phase when spans and pixels are incorporated into the workload model. In this case, the maximum speedup increases to 11.9 on 16 processors, which is more than twice the maximum speedup when only the number of triangles is considered. Figure 14 illustrates the speedup values when the execution times of the decomposition algorithms are included in the running times. Comparison of Figures 13 and 14 shows that the decomposition overhead does not introduce substantial performance degradation in the overall parallel algorithm. For example, the maximum speedup on 16 processors slightly reduces from 11.9 to 10.7.

As seen in Figure 14, the best speedup values are achieved by the 1D horizontal decomposition (HD) scheme. This is an unexpected result since HD has the worst load-balancing and primitive-replication performance. However, HD has an advantage over the other decomposition schemes for Challinger's DVR algorithm which heavily exploits both inter- and intra-scanline coherency. HD preserves intra-scanline coherency since screen is not divided vertically. However, the 2D decomposition schemes disturb both types of coherency. In addition, the bounding-box approximation used for estimating the span and pixel counts of the regions is likely to introduce more errors when screen is divided both horizontally and vertically

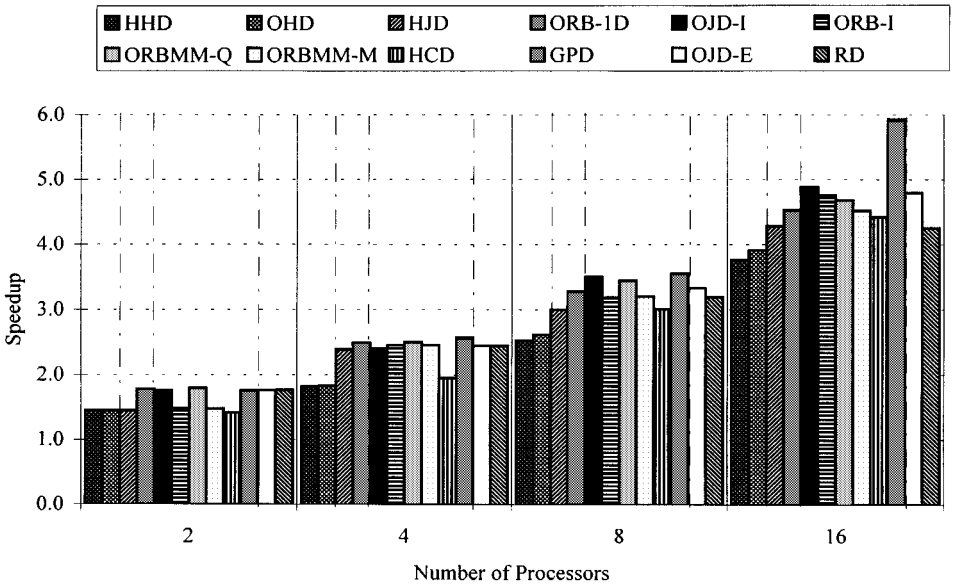


Figure 12. Speedup for the parallel rendering phase when only the number of triangles is used to approximate the workload in a region.

than it is divided only horizontally. As mentioned earlier (see Figure 1), the span counts of the regions are estimated accurately when only horizontal division lines are allowed. However, when vertical divisions are also allowed, the bounding-box approximation introduces errors in the estimation of the span counts of the regions. In spite of these findings, the 2D decomposition algorithms are expected to yield better parallel performance for larger number of processors due to their much better load-balancing and primitive-replication performances.

As seen in Figures 13 and 14, the speedup values are not very close to linear. This experimental finding stems from the errors in estimating the workload associated with a screen region. The number of spans and pixels to be generated by a triangle are calculated erroneously because of the bounding-box approximation. The second source of errors comes from the errors in the experimental estimation of the unit computational costs of processing a triangle, a span and a pixel (i.e., constants α_T, α_S, and α_P in Eq. (1)). These unit costs can not be determined precisely through measurement because of the highly interleaved execution manner of the respective types of computations.

10. Conclusions

Several adaptive image-space decomposition algorithms were presented according to a novel taxonomy for sort-first parallelism in direct volume rendering of unstructured grids on distributed-memory architectures. The proposed taxonomy is based on the dimension of the screen decomposition and the dimension of the workload

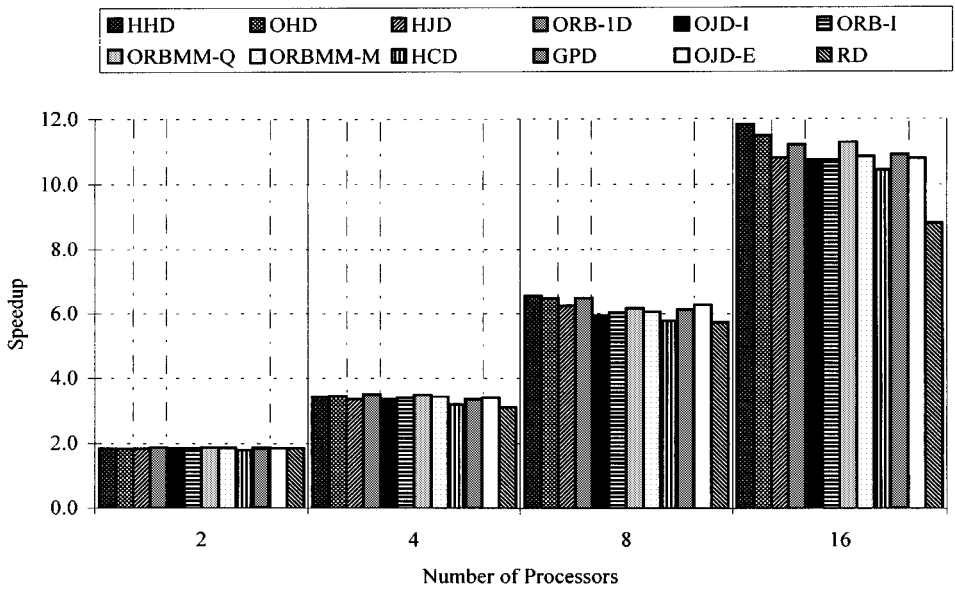


Figure 13. Speedup for the parallel rendering phase when spans and pixels are incorporated into the workload model.

arrays used in the decomposition. The decomposition algorithms were parallelized as much as possible to reduce the preprocessing overhead.

The screen-space bounding boxes of the primitives were used to approximate the coverage of the primitives on the screen. The number of bounding boxes in a screen region was used as the workload of the region. For the 2D decomposition schemes using 2D workload arrays, a novel scheme was proposed to query the exact number of bounding boxes in constant time, whereas the inverse area heuristic (IAH) was used in the literature for estimating the workload of a region.

The chains-on-chains partitioning (CCP) algorithms were exploited for load balancing in some of the proposed decomposition schemes. The probe-based CCP algorithms were used for optimal 1D horizontal decomposition and iterative 2D rectilinear decomposition. A new probe-based algorithm was proposed for finding globally optimum 2D jagged decompositions of general workload arrays. The summed-area table (SAT) scheme, which allows to find the workload of any rectangular region in constant time, was exploited to reduce both the run-time efficiency and computational complexity of the 2D optimal jagged and iterative rectilinear decomposition of general 2D workload arrays. New 2D decomposition algorithms using the IAH model were implemented for image-space decomposition. The orthogonal recursive bisection approach with the medians-of-medians scheme was applied on regular mesh and quadtree superimposed on the screen. The Hilbert space-filling curve was exploited for image-space decomposition. A graph-partitioning based decomposition scheme was also proposed and implemented. An efficient primitive classification scheme was proposed for redistribution in 1D horizontal, 2D rectilinear and 2D jagged decompositions. The complexity of the proposed classification

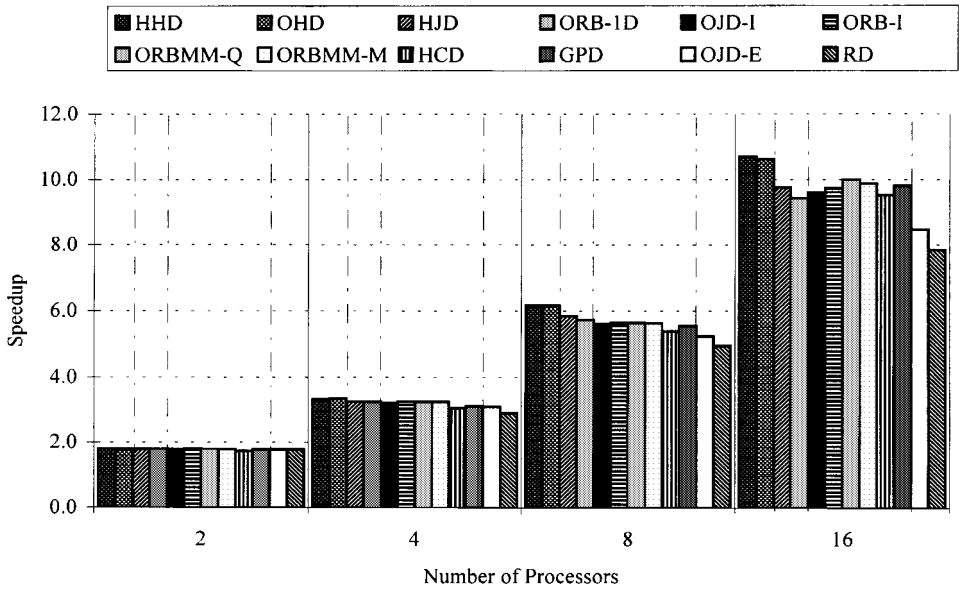


Figure 14. Speedup for the overall parallel rendering algorithm (including decomposition time) when spans and pixels are incorporated into the workload model.

scheme per primitive is independent of both the number of processors and the screen resolution.

The load-balancing, primitive-replication and parallel run-time performances of the decomposition algorithms were compared both theoretically and experimentally. The theoretical models for the comparison of load-balancing and primitive-replication performances were based on establishing appropriate quality measures. The experimental results on a Parsytec CC system using a set of benchmark volumetric datasets verified the validity of the proposed quality measures. The following two topological properties of the underlying decomposition scheme have a very strong effect on the amount of primitive replication: the total perimeter of the regions and the total number of junction points generated with the division lines. The 2D orthogonal recursive bisection (ORB) scheme utilizing the exact workload model achieves the best primitive-replication performance, and the 2D jagged and rectilinear schemes utilizing the exact model display close performance to that of ORB. The amount of primitive replication has a very strong effect on the load-balancing performance of a decomposition algorithm. The 2D decomposition algorithms achieve substantially better load-balancing performance than the 1D algorithms since the 2D algorithms have larger solution space and incur less amount of primitive replication. The optimal jagged decomposition through using the exact model achieves the best load-balancing performance, and the ORB scheme utilizing the exact model gives the second best performance. The decomposition algorithms that use 1D workload arrays run faster than the 2D decomposition algorithms that use 2D workload arrays. The performance evaluation of the presented image-space

decomposition algorithms was also carried out through the sort-first parallelization of an efficient DVR algorithm.

The presented decomposition algorithms can be extended to the decomposition of 2D and 3D nonuniform workload arrays for the parallelization of irregular and loosely synchronous data-parallel applications such as molecular dynamics, sparse matrix computations, image processing, and FEM and CFD simulations.

References

1. C. Aykanat, V. İşler, and B. Özgüç. Efficient parallel spatial subdivision algorithm for object-based parallel ray tracing. *Computer-Aided Design*, 26(12):883–890, 1994.
2. S.H. Bokhari. On the mapping problem. *IEEE Trans. Computers*, 3:207–214, 1981.
3. J. Challinger. Parallel volume rendering for curvilinear volumes. In *Proceedings of the Scalable High Performance Computing Conference*, pp. 14–21. IEEE Computer Society Press, April 1992.
4. J. Challinger. *Scalable Parallel Direct Volume Rendering for Nonrectilinear Computational Grids*. Ph.D. thesis, University of California, 1993.
5. J. Challinger. Scalable parallel volume raycasting for nonrectilinear computational grids. In *Proceedings of the 1993 Parallel Rendering Symposium*, pp. 81–88. IEEE Computer Society Press, October 1993.
6. F.C. Crow. Summed-area tables for texture mapping. *Computer Graphics*, 18(3):207–212, 1984.
7. D. Ellsworth. A multicomputer polygon rendering algorithm for interactive applications. In *Proceedings of the 1993 Parallel Rendering Symposium*, pp. 43–48. IEEE Computer Society Press, October 1993.
8. M.P. Garrity. Raytracing irregular volume data. *Computer Graphics*, 24(5):35–40, 1990.
9. M. Grigni and F. Manne. On the complexity of the generalized block distribution. *Lecture Notes in Computer Science*, 1117:319–326, 1996.
10. G. Karypis and V. Kumar. MeTiS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing prderings for sparse matrices version 3.0. University of Minnesota, Department of Computer Science, Army HPC Research Center, Minneapolis, 1998.
11. T.M. Kurç, H. Kutluca, C. Aykanat, and B. Özgüç. A comparison of spatial subdivision algorithms for sort-first rendering. *Lecture Notes in Computer Science*, 1225:137–146, 1997.
12. Tahsin M. Kurç. Parallel rendering on multicomputers. Ph.D. thesis, Bilkent University, Computer Engineering Department, 1997.
13. H. Kutluca. Image-space decomposition algorithms for sort-first parallel volume rendering of unstructured grids. MS thesis, Bilkent University, Computer Engineering Department, 1997.
14. H. Kutluca, T.M. Kurç, and C. Aykanat. Experimenting with the communication performance of Parsytec CC system. Technical report BU-CEIS-9811. Computer Engineering Department, Bilkent University, 1998.
15. K. Ma. Parallel volume ray-casting for unstructured-grid data on distributed-memory multicomputers. In *Proceedings of 1995 Parallel Rendering Symposium*, pp. 23–30, October 1995.
16. F. Manne and T. Sørevik. Optimal partitioning of sequences. *J. Algorithms*, 19:235–249, 1995.
17. F. Manne and T. Sørevik. Partitioning an array onto a mesh of processors. In *Procedings of the 3rd International Workshop on Applied Parallel Computing (PARA'96)*, pp. 467–476, 1996.
18. S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, July 1994.
19. B. Moon, H.V. Jagadish, C. Faloutsos, and J.H. Saltz. Analysis of the clustering properties of Hilbert space-filling curve. Technical report UMCP-CSD:CS-TR-3611. UMIACS, University of Maryland at College Park, 1996.
20. C. Mueller. The sort-first rendering architecture for high-performance graphics. In *Proceedings of 1995 Symposium on Interactive 3D Graphics*, pp. 75–84, 1995.
21. D.M. Nicol. Rectilinear partitioning of irregular data parallel computations. *Journal of Parallel and Disributed Computing*, 23:119–134, 1994.

22. *Embedded Parix (EPX) ver. 1.9.2 User's Guide and Programmers Reference Manual*. Parsytec GmbH, Germany, 1996.
23. J.R. Pilkington and S.B. Baden. Dynamic partitioning of non-uniform structured workloads with space-filling curves. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):288–299, 1996.
24. F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.
25. D.R. Roble. A load balanced parallel scanline z-buffer algorithm for the ipsc hypercube. In *Proceedings of Pixim'88*, pp. 177–192, Paris, France, October 1988.
26. P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, 1990.
27. J.P. Singh, C. Holt, J.L. Hennessy, and A. Gupta. A parallel adaptive fast multipole method. In *Proceedings of Supercomputing 93*, pp. 54–65, 1993.
28. S. Whitman. *Multiprocessor Methods for Computer Graphics Rendering*. Jones and Bartlett Publishers, 1992.
29. P. L. Williams. em Interactive direct volume rendering of curvilinear and unstructured data. Ph.D. thesis, University of Illinois at Urbana-Champaign, 1992.
30. P.L. Williams. Visibility ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, 1992.
31. R. Yagel and R. Machiraju. Data-parallel volume rendering algorithms. *Visual Computer*, 11: 319–338, 1995.