6 The Conveyor - an Interconnection Device for Parallel Volumetric Transformations

Daniel Cohen
Reuven Bakalash

ABSTRACT This paper presents the conveyor, an interconnection device which operates on a 3D skewed memory space and provides the capability of parallel volumetric transformation. The special concept of the conveyor, its design and implementation are discussed.

6.1 Introduction

Volumetric graphics is an emerging alternative approach to the traditional 3D surface graphics [2]. The volumetric scene is represented in a discrete fashion employing a 3D lattice of voxels (3D raster). One can assume that the voxel space is an $N \times N \times N$ array, where N defines the resolution. A typical voxel space occupies a large amount of memory, for example, a moderate resolution of N equal to 512, the 3D raster consists of more than 100 million of voxels. Although such large memories are becoming feasible, the extremely large throughput that has to be supported requires special architectural and processing solutions. In order to achieve real-time performance, parallel access and processing are essential [3]. Typically, voxel space operations, such as projections, 3D transformations (rotation, translation, shearing etc.), 3D filtering or synthesis, have the complexity of $O(N^3)$, which is linear to the number of voxels [4].

In the approach we have taken the volumetric operations mentioned above are unified and defined on 3D windows called *rooms*. A room is a box aligned with the main axes. The room can, thus, be treated as a 2D array of *beams*, where a beam is defined as a sequence of consecutive voxels parallel to one of the main axes. It is therefore desirable to have a mechanism that provides parallel access and displacement of beams. This mechanism can support volumetric operations to that are performed on a beam by beam basis, and therefore reduces the time complexity by an order of magnitude. That is, the complexity is $O(N^2)$, linear to the number of beams, rather than linear to the number of voxels. The capability to read and write an arbitrary beam of any size is, therefore, the key for real-time volumetric operations and it plays a central role in the Cube [5] and the Flipping Cube [6] architectures.

The design of parallel memories that permit conflict-free access to rows and columns is widely known [7, 8]. The extension to 3D memories is described shortly in Section 6.2, where we propose the *conveyor* - a mechanism to displace a beam of arbitrary size and orientation from an arbitrary source location to an arbitrary destination in few memory cycles. The challenge confronting the realization of the conveyor is efficient displacements by arbitrary distance in a constant time. Moreover, it has to deal with very long beams (512 or more), and it must be feasible for VLSI realization. In Section 6.4 we describe the special hardware implementation of the *conveyor*.

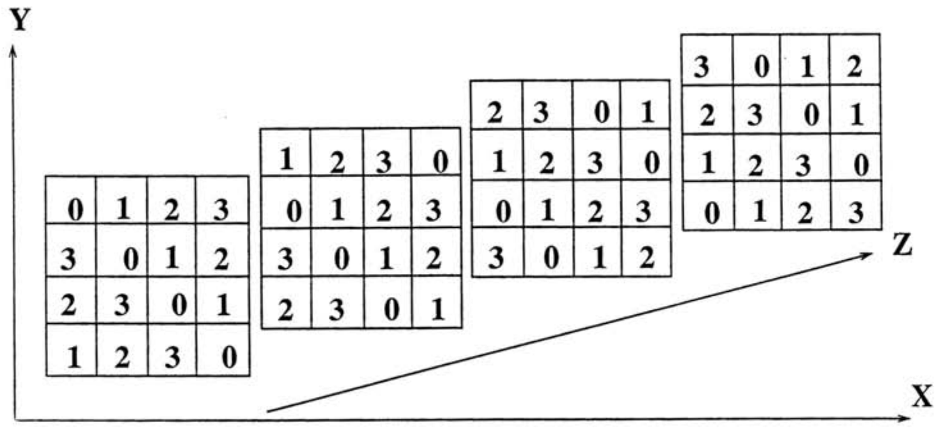

FIGURE 6.1. A latin cube of order 4.

6.2 Conflict-Free Access to Beams

A *Latin square* of order N is an $N \times N$ array of symbols from 0 to $N-1$ such that no symbol appears more than once in any row or any column. The Latin square provides a skewing scheme for the partition of the 2D space into memory modules in way that guarantee conflict-free access to 2D beams.

A linear skewing scheme is a simple mapping K from the x, y space to the N memories modules, defined by:

$$K(x,y) = a * x + b * y \,(mod\, N) \tag{6.1}$$

if $gcd(a,N) = 1$ and $gcd(b,N) = 1$ then K defines mapping into a latin square. Similarly a 3D linear skewing scheme

$$K(x,y,z) = a * x + b * y + c * z \,(mod\, N) \tag{6.2}$$

such that *gcd* of $(a,N), (b,N)$ and (c,N) are all equal to 1, defines a mapping into a *latin cube*, which provides a conflict-free access to 3D beams. In the Cube architecture the simplest 3D linear skewing scheme

$$K(x,y,z) = x + y + z \,(mod\, N) \tag{6.3}$$

has been employed [5]. This skewing scheme can be intuitively understood as a memory space which consists of diagonal parallel sections having a 45 degree angle with the main axes planes. The diagonal sections are sequentially numbered and grouped (modulo N)

into N modules indexed from 0 through $N-1$. Figure 6.1 depicts such a skewing scheme for N=4. Note that Equation 6.3 is symmetric to all the three dimensions, thus it equally treats all beams independently of their orientation.

In the following discussion the skewing scheme (3) is assumed and all arithmetics must be taken modulo N. Any beam of length N can be represented as by a sequence $i, i+1, i+2, ... i+N-1$ of memory modules, where i identifies the beam. An x-beam $X(x_0, x_1)$ from x_0 to x_1 with constant $y = y_j$ and $z = z_k$ is a sequence $s_0, s_0+1, s_0+2, ... s_1$, where

$$s_0 = x_0 + y_j + z_k \,(mod\, N) \tag{6.4}$$

and

$$s_1 = x_1 + y_j + z_k \,(mod\, N) \tag{6.5}$$

Similarly, y-beams $Y(y_0, y_1)$ and z-beams $Z(z_0, z_1)$ are defined.

An important feature of the linear skewing scheme is that two arbitrary beams of the same length are equal, up to a constant value modulo N. Let $A(a_0, a_0+l) = s_0, ..., s_l$ and $B(b_0, b_0+l) = t_0, ..., t_l$ be two arbitrary beams of length l along the dimensions A and B, respectively. For every i $(0 <= i <= l)$ $s_i = t_i + D$, where D is the *distance* between the two beams defined by

$$D = t_0 - s_0 \tag{6.6}$$

where t_0 and s_0 are calculated by Equation 6.4.

In order to displace a beam, the value of D (Equation 6.6) has to be added to all voxels along the beam. Instead of adding, they are shifted in a conveyor belt fashion within the address space. The shifting device is, therefore, termed a *conveyor*. Figure 6.2 illustrates the conceptual functionality of the conveyor that operates in three steps:

1. The selected source beam is read into the conveyor.
2. The conveyor shifts around its values to match the destination beam.
3. The shifted beam is written back to the memory.

Practically, the conveyor is an internal device that interconnects the memory modules as depicted in Figure 6.3. This flow-through mechanism provides the basic atomic operation of moving beams. In the next section we demonstrate a few 3D operations in which the complexity of room operators is reduced to those of 2D window operators.

6.3 3D Transformations Via the Conveyor

In the previous section we showed that the conveyor, by interconnecting the memory modules, provides a fast mechanism for beam transporting. One can assume that reading, shifting, and writing a beam takes a constant time. An immediate application of that mechanism is a room translation. There are three orientations by which the translation of the room can be broken into beams. Clearly it is always faster to translate beams along

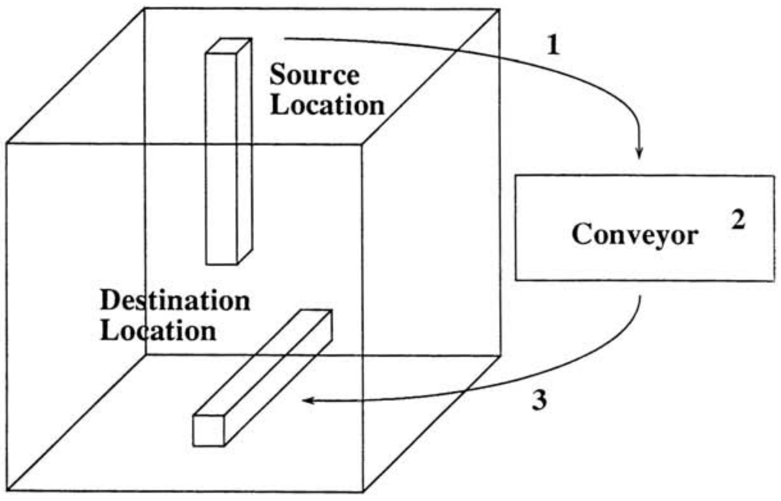

FIGURE 6.2. The concept of the conveyor: (a) a source beam is read into the conveyor. (b) The conveyor rotates the beam. (c) The beam is written into the destination location.

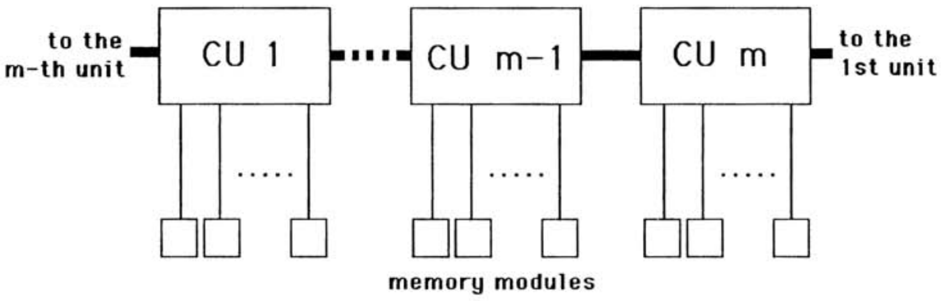

FIGURE 6.3. The conveyor interconnecting the memory modules.

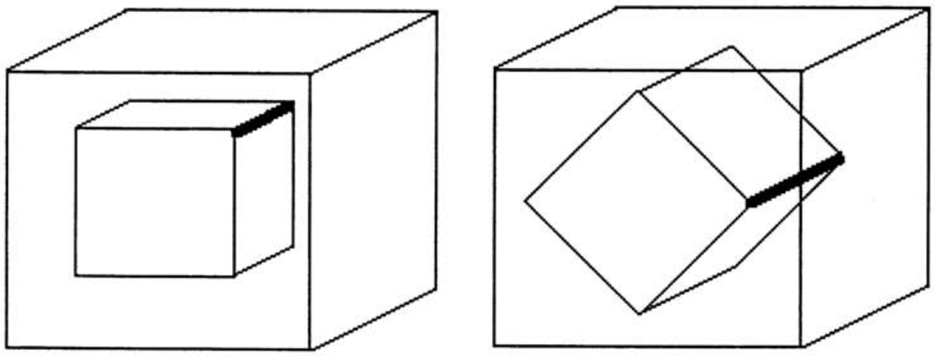

FIGURE 6.4. A beam, parallel to the main axis preserves its parallelism when rotated about the axis.

the largest dimension. Assume an $N_1 \times N_2 \times N_3$ room such that $N_3 > N_2 > N_1$, then by translating beams along N_3 the entire room can be moved in $N_1 \times N_2$ steps.

A 3D rotation is a primary application supported by the conveyor. In fact, the concept of the conveyor was first devised to support real-time rotations of volumes. The principle is shown in Figure 6.4; a rotation around one main axis is a 2D rotation where the rotation of an individual "pixel" is followed by a beam translation along the axis of rotation. A full 3D rotation is accomplished by two consecutive rotations around one axis. Note that unlike the translation operation, the parallel rotation does not have an analog 2D operation, and it is inherently a 3D feature.

Shearing is yet another 3D transformation that can be executed by the basic operation of beam translation. A sequence of three consecutive shearing operations can yield arbitrary 3D rotation. Such parallel shearing on volumes can accelerate applications of affine transformation for volume rendering [9]. In particular, a 90-degree rotation of the entire 3D space can be performed with no need for external storage, and without degradation of the image. The Flipping-cube architecture is based on that feature, and it relies on the conveyor to perform parallel shearing transformations [6].

6.4 The Conveyor Realization

The conveyor is a device capable of shifting data along N memory modules. To achieve high bandwidth, parallel shifting is necessary, while keeping a low number of installed links in order to minimize the hardware complexity. Conventional communication channels such as a single bus or ring are unacceptable due to their serial nature. Various parallel interconnection network topologies are in use in parallel processing, providing different communication patterns [10]. Two extremes are the bus and the crossbar. Multiple buses, connecting between adjacent modules have a simple topology but the performance is linear with the number of shifts. In contrast, the crossbar offers constant time connection but has N^2 link complexity. One compromise is the barrel shifter that provides an arbitrary shift in $logN$ steps but requires $N/2logN$ link complexity. Another solution is the perfect shuffle that can achieve an arbitrary shift in $logN$ steps and requires only N links.

High speed and low complexity has been achieved in the unique design of the conveyor, exploiting the fact that no two accesses in the network involve the same memory module and that there is a uniform communication distance among all units. The design of the physical device has been guided by following requirements: low hardware complexity, high bandwidth, modularity for a flexible length of network and a flexible width of data, and suitability for VLSI implementation.

Let us assume a case study of a 256^3 voxel space, where the requirement is a simultaneous shifting of 8 bits of data in an arbitrary distance among the 256 modules. Applying the existing methods it would require either 256 buses or a large crossbar matrix of 256×256 paths, 8 bit each. The conveyor, shown in Figure 6.3, is a modular structured network interconnecting among the clients (either processors or memory modules) [1]. The basic component is a *conveyor unit* (CU) which serves s clients, receiving from each a single bit of data (s bits in total) and shifting it further on. This layout provides modularity and flexibility of the entire network. The network is extendable in its overall length, by serially connecting arbitrary numbers of CU's, and in data width, by stacking up any number of units, one layer per data bit. Such a modular approach results in breaking down the entire communication network into small building-blocks easily implementable in VLSI.

The CU is a flow-through bidirectional shifter performing a shift of up to s steps. These bidirectional shifts contribute to a significant reduction in communication area and time. Rotation to the right by k steps, when $k > 256/2$, can be performed with leftward rotation of $(256 - k)$ steps.

Long rotations of k steps are subdivided into smaller flow-through s steps. A rotational distance of up to s steps is performed in a single clock. Longer rotations require k/s clocks, and the longest is performed in $N/2s$ clocks. Since, for small N, the term $N/2s$ for relatively small s is the same as $logN$, the conveyor performance equals or surpasses the performance of the barrel shifter and the perfect shuffle.

The topology of the conveyor is simple since the links connect between adjacent clients only. The CU is composed of s columns (see Figure 6.6), exhibiting link complexity of $O(s)$. Since the CU's are serially (linearly) composed, the entire network has the same $O(s)$ hardware complexity. There is a trade-off between the hardware complexity and the time performance. The bigger the step size s, the faster the network, but on the other hand, the link complexity of the CU component is increasing.

A CU is composed of three main parts (see Figure 6.6): the input-output terminals; the shift-array (SA) of communication links; and the local control sub-unit called decoder. All the signals, controlling this module, are furnished by an external control.

Data transfer through the conveyor is performed in the following manner. First, data from all the clients are moved to their associated CU via the D terminals. Next, the data is shifted out to the next CU through, for example, the R terminals, and new data is

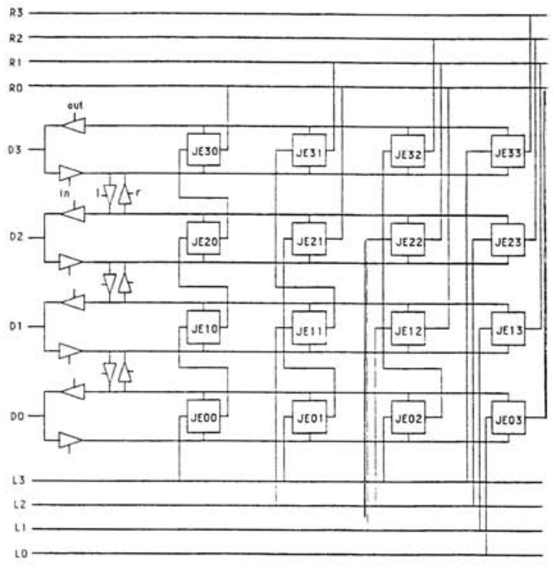

FIGURE 6.5. A simplified Shift Array (SA) with four columns ($s = 4$).

shifted in through the L terminals. A full shift, up to s places, is performed in a single clock cycle. This shifting step can be repeated any number of times. Finally, the shifted data is returned to the clients through the D terminals. The shift array is a matrix of s columns by s rows, organized in a simple and repetitive pattern of junction elements and connection paths. The data from, and to the clients are placed on a row and transferred out to the appropriate column, depending on the required shift. Each of the columns performs its unique shift distance. For example, a 3 step shift all D inputs are passed to the third column which shifts them three steps ahead and the result is placed back on D terminals for output. The decoder sets up the appropriate connections in the shift array.

The conveyor has been implemented in SUNY at Stony Brook as part of the Cube architecture. In [1] it was suggested to use a shift array with $s = 16$ for 256 modules. Rotations up to 16 consecutive locations are performed within a single clock period, and the longest rotation of 127 is executed in 8 clocks. Each CU is implemented as a custom-designed VLSI chip. The shift array contains about 2500 transistors. Adding the decoder, control circuits, and I/O pads, it totals to 4000 transistors and 64 pins per chip.

6.5 Summary

We have introduced a new technique to parallelize volumetric transformations. It relies on the ability to simultaneously read from a memory, circularly shift, and write back an

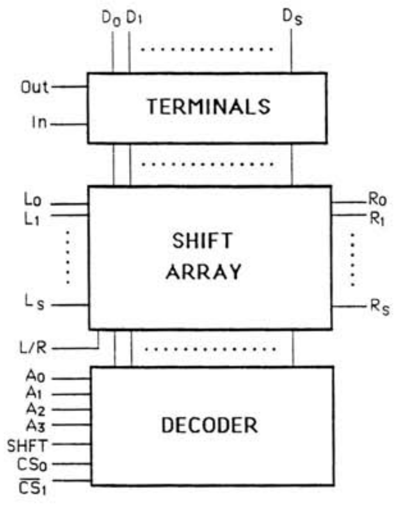

FIGURE 6.6. The Conveyor Unit (CU).

entire group of data. We have presented the conveyor, which is a shift device with a simple topology, low link complexity, high bandwidth, and is suitable for VLSI implementation.

Acknowledgements:

This work has been partially supported by the National Science Foundation under grants MIP-8805130, CDA-8947756 and IRI-9008109, and grants from Hewlett Packard and Silicon Graphics. We are grateful to Professor Arie Kaufman for his support and advise. We also wish to acknowledge the careful review and helpful comments from our colleague Roni Yagel.

6.6 References

[1] R. Bakalash and X. Zhang, Barrel Shift Microsystem for Parallel Processing. In *Proceedings Micro 23, 23rd Symposium and Workshop on Microprogramming and Microarchitecture*, Orlando, FA, November 1990.

[2] D. Cohen, A. Kaufman and R. Yagel, Volumetric Graphics. Technical Report TR 91.01.30, Department of Computer Science, SUNY at Stony Brook, January, 1991.

[3] A. Kaufman, R. Bakalash, D. Cohen, and R. Yagel, Architectures for Volume Rendering – Survey. *IEEE Engineering in Medicine and Biology*, 9(4):18–23, December 1990.

[4] A. Kaufman, The *voxblt* Engine: A Voxel Frame Buffer Processor. In: A.A.M.Kuijk (Ed.): *Advances in Computer Graphics Hardware III*, EurographicSeminars. Springer-Verlag, Berlin, 1991, p.85-102.

[5] A. Kaufman and R. Bakalash, Memory and Processing Architecture for 3-D Voxel-Based Imagery. *IEEE Computer Graphics & Applications*, 8(11):10–23, November 1988.

[6] R. Yagel, The Flipping Cube Architecture for Real Time Volumetric Graphics, This volume, Chapter 7.

[7] P. Budnik, J. Kuck, Organization and Use of Parallel Memories, *IEEE Transactions on Computers*, c-20, pages 1566-1569, 1971.

[8] H.A.G. Wijshoff, J. van Leeuwen, The Structure of Periodic Storage Schemes for Parallel Memories, *IEEE Transactions on Computers*, c-34, pages 501-505, 1985.

[9] P. Hanrahan, Three-Pass Affine Transforms for Volume Rendering, *Computer Graphics*, 24(5), pages 71-78, 1990.

[10] H.S. Stone, In *High-Performance Computer Architecture* Addison Wesley, 1987.