DLoVe: Using Constraints to Allow Parallel Processing in Multi-User Virtual Reality

Leonidas Deligiannidis
Department of Electrical Engineering and Computer Science
Tufts University, MA
ldeligia@eecs.tufts.edu

Robert J.K. Jacob
Department of Electrical Engineering and Computer Science
Tufts University, MA
jacob@eecs.tufts.edu

Abstract

In this paper, we introduce DLoVe, a new paradigm for designing and implementing distributed and non-distributed virtual reality applications, using one-way constraints. DLoVe allows programs written in its framework to be executed on multiple computers for improved performance. It also allows easy specification and implementation of multi-user interfaces. DLoVe hides all the networking aspects of message passing among the machines in the distributed environment and performs the needed network optimizations. As a result, a user of DLoVe does not need to understand parallel and distributed programming to use the system; he or she needs only be able to use the serial version of the user interface description language. Parallelizing the computation is performed by DLoVe, without modifying the interface description.

1. Introduction

We have developed a software model and user interface description language (UIDL) for programming interactive behaviors in virtual reality (VR) and other non-WIMP interfaces [31]. In it, we introduced a language element that is essentially a set of one-way constraints or a dataflow graph. We chose it for its expressive power and its suitability for describing continuous interaction. We found that, although constraints are often viewed as introducing performance penalties compared to conventional coding, our approach allowed us to improve performance or interactive responsiveness. This is because our constraint-based formalism allows a separation of concerns between the desired interactive behavior and the implementation mechanism. The user interface designer can thereby concentrate on and express the former, in a high-level, declarative, continuous-oriented way, while our underlying runtime system can perform optimization, tradeoffs, conversion into discrete steps, and allow parallel execution independently, below the level of the UIDL.

This paper introduces the DLoVe (Distributed Links over Variables evaluation) system, which provides a UIDL for programming non-WIMP interactions and a mechanismfor executing UIDL programs, designed for a single machine, on a set of distributed machines where updates on DLoVe variables can be executed in parallel for performance improvement. DLoVe also provides, as a byproduct, the framework for writing multi-user VR programs or transforming existing single-user DLoVe programs into multi-user ones. DLoVe addresses issues of performance and maintainability, providing mechanisms, drivers, and utilities that allow run time tuning and network management to be specified in a simple manner [16].

2. Background

Most of today's Graphical User Interfaces (GUI) and toolkits are based on serial, discrete, token-based paradigms that implement traditional WIMP (Window, Icon, Menu, Pointer) interfaces acceptably. These tools however, are not suited for next generation, non-WIMP interaction styles such as Virtual Reality. The fundamental characteristics of these non-WIMP interfaces are common to a more general class of emerging user-computer environments, including new types of games, musical accompaniment systems, intelligent agent interfaces, interactive entertainment media, pen-based interfaces, eye movement-based interfaces, and ubiquitous computing [29] [12]. They share a higher degree of interactivity than previous interfaces: continuous input/output exchanges occurring in parallel, rather than one single-thread, discrete event dialogue. These interaction techniques rely upon asynchronous, parallel, and continuous user/computer interaction [20].

We want to introduce a higher level, cleaner user interface description languages into this field, without

compromising performance. Constraints are often viewed as effective and expressive but too inefficient for a high-performance application like virtual reality. However, using a declarative constraint specification rather than a procedural program gives us the freedom to implement more sophisticated management of execution time while separating this concern from those of the user interface designer. It allows us to tailor the response speeds of different elements of the user interface to maximize the subjective sense of interactive responsiveness within the available computing resources. All this is specified separately from the user interface description; the UIDL constraint specification lets the designer describe only the ideal desired behavior (as if infinite computing resources were available on a single CPU).

Because the calculations are specified as a constraint graph, DLoVe can parallelize the computation to take advantage of multiple computers to improve performance. It also provides a framework to execute the same VR application in a multi-user environment.

3. Specification Language

DLoVe is based on the PMIW specification language [31], which uses a two-component model for describing and programming the fine-grained aspects of non-WIMP or VR interactions. First, we identify the basic structure or syntax of non-WIMP interaction as the user sees it. We posit that essence of the sequence of interactions in a non-WIMP interface is a set of continuous relationships, most of which are temporary. For example, in a virtual environment, a user may be able to grasp, move, and release an object. The hand and object positions are thus related by a continuous function (say, an identity mapping between the two 3-D positions)--but only while the user is grasping the object. This leads to a two-part model of user interaction. One part is a graph of functional relationships among continuous variables, much like a constraint graph, but only a few of these relationships are typically active at one moment. The other part is a set of discrete Event Handlers that can, among other actions, cause specific continuous relationships to be activated or deactivated. Most other UIDLs and software systems are based on serial, discrete, token-based interaction. DLoVe is designed to provide a fundamentally continuous, rather than discrete, treatment of naturally continuous phenomena such as time and motion. In addition, it handles discrete events as discrete events and it provides a mechanism for communication between the continuous and the discrete sub-system. Our UIDL is described in more detail and with more examples in [28] and [31]; and a related, more powerful system based on it, in [1].

4. Continuous Time

We can now observe that the continuous portion of our two-part formalism looks much like a set of one-way constraints. We believe that constraints are indeed a useful element of a UIDL for developing virtual reality user interfaces because of their expressive power [5]. They also have the benefit of separating programming of the desired behavior from programming the implementation engine; this gives us the hook to address VR performance issues. Because we use constraints, the user simply provides a declarative specification of the desired relationships and, if applicable, indicates how they are turned on and off. The underlying system does not need to guarantee that it will always update the constraints in lockstep fashion, but it will have the freedom to manage the available computer time in different ways to achieve good interactive responsiveness, without explicit effort on the part of the writer of the constraints. Our constraint solver is free to handle the task sequentially or in parallel; it might also approximate the desired behavior more or less closely, depending on the CPU time available on each frame.

DLoVe's continuous time sub-system consists of object elements that define the relationships between variables. The entire set of these elements connected together form a constraint-like graph. Changes at one end of the graph propagate to the other end. Interaction that is conceptually continuous is encoded directly into these elements, and thus the application does not need to deal with tracking events from conceptually continuous devices. Examples of conceptually continuous interaction include, drinking from a cup in a virtual world, throwing a ball in a virtual park, and driving a car in a virtual city. Many non-WIMP interfaces must meet severe performance requirements in order to maintain their perceptual illusions. For virtual reality, in particular, these requirements are the driving force behind the design of most current implementations [13]. We found that using one-way constraints in our UIDL gave us flexibility to manipulate execution, scheduling, and parallel processing on the fly and improve the interactive responsiveness of our system over that of the equivalent hard-coded system sans constraints. Our DLoVe solver allows the constraint graph to be distributed over several workstations, for faster response time as well as for supporting multi-user virtual environments.

The demand-driven or lazy evaluation algorithm in DLoVe is based on the optimal algorithm of Hudson's Eval/Vite system [32] [34]. DLoVe adds some features designed specifically for the requirements of VR interfaces. In addition, DLoVe is designed for continuous, rapidly updated response to user inputs. One consequence is that it is often reasonable to discard

queued-up inputs in favor of responding to the latest input. Another is that many of the most rapidly-changing inputs (the 3-D trackers) are fairly well behaved because they are generated by the user's limb movements.

Continuous time in DLoVe is handled via Variables and Links. Variables are objects in DLoVe that store values and know which Links need them as inputs and which for output. They are invariant data flow graph elements that serve as both continuous and short-term data repositories. Some Variables are directly connected to input devices, some to outputs, and some to application semantics. Some are used for communication within the user interface model and some just hold intermediate results of Link calculations.

Links are objects that contain functions and are attached at both ends to Variables. Links get input from Variables and place the result of their calculations into other Variables. The body of a Link specifies how the attached Variables are related. Links can be enabled or disabled in response to user inputs. When a Link is disabled, it is as if this Link were not part of the constraint network any more. By enabling and disabling Links, we can quickly change the constraint network on the fly since only a flag needs to be set or cleared to indicate that a Link is enabled or disabled. This ability to enable and disable portions of the data-flow graph in response to user inputs is a key feature of the model. Links can have multiple outputs and multiple inputs. In other cases, a Link may read input Variables and perform some complex function on them over time, which eventually generates a token for the Event Handlers. This can be useful for gesture recognition or for eye tracking applications where, for example, when the user looks at an object for a certain time, the object becomes selected. We developed an application where, when the user looks toward an object in a virtual world for over 5 seconds, the object becomes selected [16]. The brightness of the object is proportional to the time for which the object has been viewed, and when it reaches a certain brightness level, the object is selected and the user can manipulate it. In addition, a single Variable may be used as input or output to multiple Links. Conditions are also provided to enable and disable groups of Links instead of enabling or disabling Links individually.

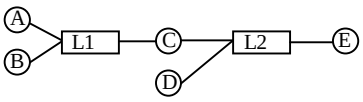

Figure 1. Links and Variables

Figure 1 above shows the Variables as circles and the Links as rectangles. When a Link is created, it is enabled by default. In the DLoVe constraint graph we draw a crossed circle on top of a Link to indicate that is disabled. A DLoVe graph is read from left to right. For example, Variables 'A' and 'B' are inputs to Link L1, and Variable 'C' is its output Variable. When L1 is disabled it is as if this Link was deleted from the network. However, a disabled Link is still part of the data structure and when it becomes enabled again it knows how it is supposed to be attached to its Variables. The relationship between 'A', 'B' and 'C' is terminated temporarily until 'L1' becomes enabled again.

The programmer specifies the input and output Variables to each Link. This is represented as a data-flow graph that defines which output Variables will be affected by a change in the input Variables. DLoVe then automatically generates a dependency graph where, for each output Variable, all relevant input Variables can be found by 'breadth-first' search. This enables DLoVe's incremental constraint algorithm to work efficiently starting at the requested Variables and working backwards to where the change occurred, and start evaluating Links in topological order, if needed.

5. Discrete Time

There are other however, interactions that are fundamentally discrete (event-based) and, for them, DLoVe provides an event-based component. Such examples include button presses, menu choices and gesture recognition verifications. They often result in enabling or disabling constraints in the graph; for example, TBAG applications [8] generally deal with such discrete input events by retracting some existing constraints and asserting new ones. Bramble uses a similar mechanism [19].

DLoVe handles the discrete time using Event Handlers, objects that capture tokens and respond to them. DLoVe reads all hardware input events from the X Window system and any other sources and turns them into tokens and sends them to all Event Handlers. Event Handlers contain a user-specified body that describes the response to tokens. The application sends a token to all Event Handlers, and only those Event Handlers that are interested in the token execute their bodies. The responses might include setting Variables, making custom procedure calls, and enabling or disabling Links. Event Handlers recognize states and state transitions, and can provide different services depending on the state they are in.

In an example of a factory with moving objects on its assembly line, an Event Handler state diagram might look like figure 2 (where the communication between the Event Handler and the continuous time subsystem is done by Enabling and Disabling the L1 and L2 Links):

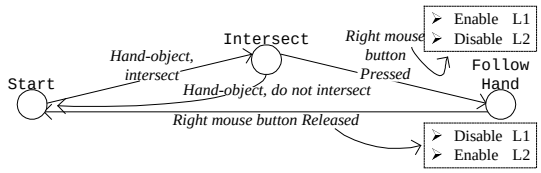

Figure 2. State Diagram of the Event Handler

When the user's hand intersects with the moving object, the Event Handler receives a token (e.g. "ENTER"), the object becomes highlighted, and the Event Handler transitions to the "Intersect" state. At this state, if the user presses the right mouse button, the Event Handler receives another token (e.g. "LEFTDN"), transitions to the "Follow Hand" state, enables Link L1, and disables Link L2. Now the hand-object relation has been established, and the object follows the movement of the hand. When the mouse button is released, the Event Handler receives another token (e.g. "LEFTUP"), transitions to the "Start" state, disables Link L1, and enables Link L2. At this point, the hand-object relationship is terminated. Further details of this state diagram notation itself are found in [30] [27].

6. Parallel Processing in DLoVe

One of the major difficulties with parallel processing is in transforming a program written for a sequential machine into one that executes on multiple machines in parallel. A program written in DLoVe, however, can be initially written to execute on a single machine sequentially. The same program can then execute in parallel on multiple workstations with hardly any source code changes. The only difference between the parallel and serial versions is that different libraries are used to compile the executables. When compiling for parallel execution, compilation generates two different executables, one for the Coordinator (the machine responsible for reading input devices and rendering graphics), and another for one or more Workers (the machines responsible for doing constraint calculations in parallel). The Coordinator is a workstation with a display device and input devices, such as polhemus 3D tracker, mouse, and eye tracker. It is responsible for reading all data from the input devices and rendering the graphics. Workers are only responsible for doing calculations based on the requests from the Coordinator.

In our approach to parallel processing, each Worker and the Coordinator own an exact copy of the constraint graph. Each Worker can execute the constraint solver on any given Variable. However, at the initialization phase the Coordinator partitions the graph so that it can request certain Variables from certain Workers. In essence, the Coordinator assigns 'tasks' to individual Workers. The Coordinator partitions the graph so that related Variables form a group of Variables in the Workers, and requests to these specific Variables are always sent from the Coordinator to the Worker that owns this portion/partition of the graph. With this approach to parallel processing, distributing the calculations over several machines cannot adversely affect correctness since all machines maintain the entire graph; it only affects response speed.

The Coordinator executes the following algorithm to partition the graph. It first finds the number of Variables upon which each Variable depends. Next, it assigns, in round-robin, the Variables with the most dependencies, recursively, to different Workers as shown below:

```
foreach Variable v in AllVariables do
   v.depend = Breadth-first-search(v);

int w = 0;
Worker worker[total_num_of_Workers];

foreach Variable v in Next_highest_depend do
   if v.assigned != assigned_to_any_worker then
      worker[w].Recursive_assign(v);
      w = (w+1) mod total_num_of_Workers;
```

Because the Coordinator also has a complete copy of the constraint graph and can execute the same constraint algorithm, it enables it to bring critical Variables up-to-date locally, when the computations involved are inexpensive. This is useful for Variables that control the position and orientation of the user's head and hand that must be updated as quickly as possible to maintain the user's sense of immersion.

The Coordinator can asynchronously send large messages, to reduce network overhead [10]. A message can contain multiple requests such as Enable or Disable a particular Link, Set a Variable to a value, and requests for updated Variables. Workers only reply to the Coordinator to requests of updated Variables in a similar manner, batching multiple results in a large message. There is no communication between the Workers themselves.

7. Multiple Users

Since our constraint graph is distributed and the Coordinator knows how to request Variables from the Workers and how to set the values of Variables in the Workers, it is a simple matter to add additional Coordinators to support a multi-user interface. In DLoVe, adding another Coordinator (i.e., another user) simply requires specifying the number of additional Coordinators in DLoVe's configuration file and running the additional

Coordinators using the same executable as the original Coordinator.

While multi-user execution is easy, there are several issues that arise in designing any multi-user interface. The main issue is that the UIDL must now describe the handling of each user's input individually; for two users, we must write a user interface description involving two different mice or two hand trackers. For example, assume that there is only one Variable, 'mouse', attached to the mouse device so that the position of the mouse represents the position of the user's virtual hand in space. In a multi-Coordinator environment, each Coordinator can set the Variable 'mouse' resulting in multiple Coordinators manipulating the single virtual hand. But a multi-user interface requires the designer to specify the interaction for *each* user's mouse; in some interface designs, different users might play different, asymmetric roles in the interaction. DLoVe assigns, in its configuration file, application layer identification numbers to each physical device attached to each machine. Each device is automatically translated into <device>_ID where device is the original name of the device, such as mouse, and ID is the application layer identification number, yielding a different name for each machine's device such as mouse_2 and mouse_6. Links attached to input devices might also need to be duplicated and rewired to handle the multiple-user interface. This allows all users in a multi-user environment to be able to control their own heads and hands and to be able to see all other users participating in the simulation. The application identification numbers also allow programmers to write programs and assign different roles and responsibilities to different users (such as pitcher and catcher). We developed an application where one user can only manipulate 'red' objects and the other user 'blue' objects. By pressing a keyboard command, the roles are reversed and the first user can only grab 'blue' objects and the second only 'red'.

Several applications have been developed using DLoVe to demonstrate its strengths and weaknesses. The Virtual Park application was developed to test performance and to demonstrate flexibility and efficiency. The Virtual Park consists of several objects such as sliders controlling orbiting objects, arms controlling the direction of other arms, and Humanoids, which are human-like entities walking, wandering, and playing with a virtual ball. The Humanoids look at the ball, avoid collisions with the other Humanoids, and when they come close to the ball, they hit it. The virtual ball can travel over the park depending on the force with which it was hit, ending finally on the ground where gravity pulled it. The user(s) can grab all objects including the Humanoids and the ball. The users can grab and toss the ball the same way the Humanoids interact with it. The behavior of each Humanoid, collision behavior, speed, etc, is based on a complex function that accounts the location and the age of all other Humanoids, much like magnets with different magnetic strengths. A snapshot of the DLoVe Virtual Park is shown in Figure 3. Part of the constrained graph is shown in Figure 4, because of space limitations, that shows the ball object and three of the Humanoids. Each Humanoid is assigned as a single task to a single Worker.

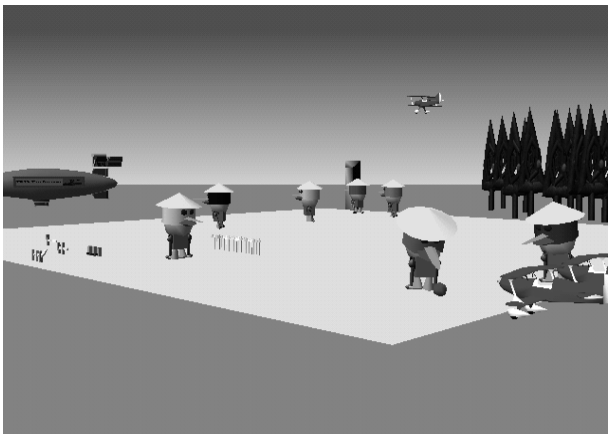

Figure 3. Virtual Park

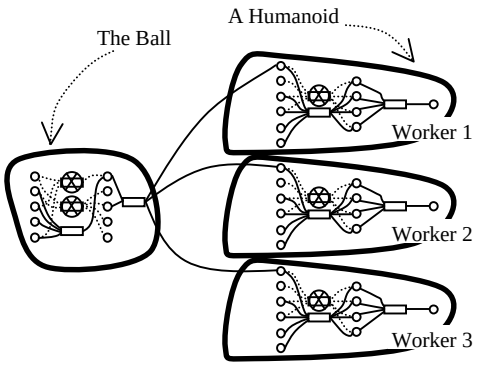

Figure 4. Constraint Graph of the Virtual Park

8. Evaluation: Performance Measurement

Our final task was to evaluate DLoVe with quantitative measurements of the performance improvement it provides through parallelism. Initial measurements suggested that conventional measures used to evaluate parallel processing systems are less useful in VR [2] [11] [7].

Table 1 shows that by using more Workers we increase the throughput of the system but we decrease the frame rate. The throughput increase is due to parallel computations performed by the multiple Workers, where the decrease in frame rate is due to the fact the DLoVe is implemented on top of TCP. The Coordinator needs to send multicasts to all Workers and thus it has to simulate multicasting [16], since it is not supported by TCP. By simulating multicasting, the Coordinator spends more

time trying to send the requests to the Workers and less time to render the display. We measured the number of evaluations of one of the Links in the Humanoids which was needed in every frame to figure out how many times a Humanoid was brought up to date.

Table 1.

Number of Workers	Number of Evaluations	Frame Rate
1	1700	20
2	3500	18
3	4500	15

Throughput, or total work performed, does not accurately capture the interactive performance of a VR user interface. The system might be doing a great deal of processing by generating many frames, but displaying each of them late. It would score well by traditional parallel processing measures, but would seem very sluggish to the user.

We therefore used total throughput along with total latency of a displayed video frame to measure performance. That is, when a frame is displayed, what is the oldest piece of data that was used to generate that frame? We believe that frame latency, in conjunction with frame rate, captures the interactive performance of a VR user interface better than total processing throughput. Frame latency shows how valid or how late the information is at the time the Coordinator renders the display.

To perform this analysis, we tracked each message through the network. Message latency shows how long a message spends on the network to get to the Workers and then to come back to Coordinator (minus the time required by a Worker to evaluate a Link). We instrumented DLoVe so the Coordinator timestamps each request before sending it to a Worker. The Worker also timestamps each request with the time it took it to run the constraint solver to bring that Variable up-to-date. When the Coordinator receives a request, it calculates the elapsed time of each request minus the time the Worker needed to bring the Variable up-to-date. This is the message latency measured for each request, that is, the time this request spent in the network. However, this does not show how accurate a rendered frame is. To visualize how valid the frames are, we used a statistical clock skew:

```
skew = (wall clock) - min(time of request of
                          all requested Variables)
```

For every frame, the minimum time of request of all output Variables is subtracted from the current time (wall clock). This skew describes the worst difference between what is rendered and what the user is doing. Figure 5 is one of many plotted that shows this (see [16] for many additional performance tests).

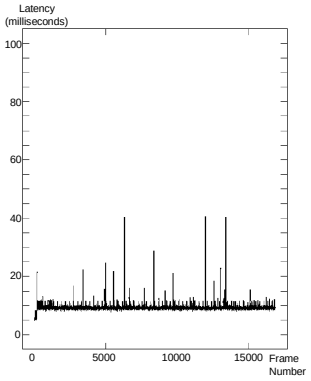

Figure 5. Worst Frame Latency

Finally, observe that these speedups are made possible by a high-level constraint-based UIDL, which placed no assumptions or restrictions on the processing sequence. Our experience with DLoVe demonstrates that introducing a higher-level, declarative user interface description language provided extra degrees of freedom to the underlying implementation. Rather than causing a performance penalty, the constraint-based language made possible the increased performance we obtained through parallel processing.

9. Related Work

Our continuous model is similar to a data-flow graph or a set of one-way constraints between actual inputs and outputs and draws on research in constraint systems [32] [34]. The model provides the ability to "re-wire" the graph from within the dialogue. Several researchers are using constraints for 2-D graphical interfaces [25] [26] [33] [6] [4]. Kaleidoscope [3] is a constraint-based language motivated by 2-D WIMP interfaces, and it explicitly supports temporary constraints.

VIVA [36] introduced some level-of-detail time management techniques in a data-driven, real-time constraint application. The CONDOR system uses a constraint or data-flow model to describe interactive 3-D graphics [21]. TBAG also uses constraints effectively for graphics and animation in the interface [8]. Gleicher provides constraints that are turned on and off by events [19]. Other recent work in 3-D interfaces uses a continuous approach [22] or a discrete, but data-driven approach [17].

Software architectures for virtual reality interfaces have been developed by Feiner and colleagues [35] and by Pausch and colleagues [18]. Green and colleagues developed a toolkit for building virtual reality systems [9]. Most of this work has thus far concentrated on the

architecture or toolkit level, rather the user interface description language. Lewis, Koved, and Ling, addressed non-WIMP interfaces with one of the first UIMSs for virtual reality using concurrent event-based dialogues [14].

Parallel Virtual Machine PVM is a message-passing software system that allows the utilization of a heterogeneous network of parallel and serial computers as a single computational resource [38]. The Message-Passing Interface (MPI) is a standard specification designed for writing distributed memory parallel processing utilizing message-passing [23] [24]. There are three main differences between DLoVe and other parallel systems. While DLoVe's tasks appear externally similar to those in PVM, task allocation is done at compile time, so that there is not appreciable overhead for task management. The purpose of task allocation in DLoVe, is to allow the Coordinator to always request the same Variables from the same Workers. In other words, the queries the Coordinator sends to the Workers are partitioned, so that the Workers can execute multiple different queries in parallel.

DLoVe's task handling, unlike PVM or MPI, is designed to support multi-user, multi-input application development. Adding a second user to DLoVe's framework, adds a second Coordinator. This means that the Workers now have to serve requests for both Coordinators making each of the Workers work harder, consume more resources, and load the network with more messages.

The third difference concerns performance requirements. DLoVe is designed primarily for Virtual Reality applications and thus requires high frame rate. Distributed applications using DLoVe's framework are characterized by real-time computations and constraints. Thus, not only number of evaluations, but also timing factors need to be taken into account when evaluating DLoVe [15].

Timing constraints in DLoVe arise from interaction requirements between the Coordinator and the user, and between the Coordinator and the Workers. The communication between the Coordinator and the Workers is described by three operations: sampling, processing, and responding. The Coordinator continuously samples data from the input devices. Sampled data is sent to the Workers that process it immediately. Then the Workers send the processed data back to the Coordinator in response to its request. All three operations must be performed within specified times; these are the timing constraints [15] [37].

10. Conclusion

DLoVe was designed to provide a specification language and execution environment for rapid, parallel execution of non-WIMP interfaces. Because it uses a high-level declarative paradigm, it allows programs to be executed in a distributed or non-distributed environment where speed is a requirement, with hardly any code modifications. It also allows easy specification of functionality for multi-user interfaces, following a simple pattern. Its run-time engine is responsible for performance optimization and network control. It hides all the networking aspects of message passing among the machines in the distributed environment. As a result, the DLoVe programmer does not need to understand distributed and parallel systems to employ DLoVe; he or she need only be familiar with the serial UIDL.

We also introduced a more useful measure of VR user interface performance than total throughput. We defined performance as the latency of the data in each frame. Our experience with DLoVe demonstrated that introducing a higher-level declarative UIDL provided extra degrees of freedom in the underlying implementation. Rather than causing a performance penalty, the constraint language made possible the increased performance we obtained with parallel processing.

11. References

[1] S. A. Morrison, "A Specification Paradigm for Design and Implementation of non-WIMP Human-Computer Interactions," Doctoral dissertation, Tufts University (1998).

[2] Amdahl, G.M. Validity of the single-processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., 1967, pp. 483-485.

[3] B. Freeman-Benson and A. Borning, "The Design and Implementation of Kaleidoscope'90, a Constraint Imperative Programming Language," Proc. IEEE Computer Society International Conference on Computer Languages, pp. 174-180, April 1992.

[4] B. Vander Zanden, B.A. Myers, D.A. Giuse, and P. Szekely, "Integrating Pointer Variables into One-Way Constraint Models," ACM Transactions on Computer-Human Interaction, vol. 1, no. 2, pp. 161-213, June 1994.

[5] B.A. Myers et al., "The Garnet Toolkit Reference Manuals: Support for highly Interactive, Graphical User Interfaces in Lisp." Tech. Report CMU-CS-90-117, Carnegie Mellon University, Computer Science Department, Mar. 1990.

[6] B.A. Myers, D.A. Giuse, R.B. Dannenberg, B. Vander Zanden, D.S. Kosbie, E. Pervin, A. Mickish, and P. Marchal, "Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces." IEEE Computer, vol. 23, no. 11, pp. 71-85, November 1990.

[7] Berman, Kenneth A. and Paul, Jerome L. "Fundamentals of Sequential and Parallel Algorithms". PWS Publishing Company, 1997.

[8] C. Elliot, G. Schechter, R. Yeung, and S. Abi-Ezzi, "TBAG: A High Level Framework for Interactive, Animated 3D Graphics

Applications," Proc. ACM SIGGRAPH'94 Conference, pp. 421-434, Addison-Wesley/ACM Press, 1994.
[9] C. Shaw, M. Green, J. Liang, and Y. Sun, "Decoupled Simulation in Virtual Reality with the MR Toolkit," ACM Transactions on Information Systems, vol. 11, no. 3, pp. 287-317, 1993.
[10] Chris Lewis, "Cisco TCP/IP Routing Professional Reference (second edition)", McGraw-Hill Companies, Inc. 1998
[11] Hesham El-Rewini & Ted G. Lewis, "Distributed and Parallel Computing". Mannining Publications Co. 1998.
[12] J. Nielsen, "Noncommand User Interfaces," Comm. ACM, vol. 36, no. 4, pp. 83-99, April 1993.
[13] J. Rohlf and J. Helman, "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics," Proc. ACM SIGGRAPH'94 Conference, pp. 381-394, Addison-Wesley/ACM Press, 1994.
[14] J.B. Lewis, L. Koved, and D.T. Ling, "Dialogue Structures for Virtual Worlds," Proc. ACM CHI'91 Human Factors in Computing Systems Conference, pp. 131-136, Addison-Wesley/ACM Press, 1991.
[15] Jeffrey J. P. Tsai, Yaodong Bi, Steve J. H. Yang, and Ross A. W. Smith, "Distributed Real-Time Systems", John Wiley & Sons, Inc. 1996.
[16] L. Deligiannidis, "DLoVe: A Specification Paradigm for Designing Distributed VR Applications for Single or Multiple Users," Doctoral dissertation, Tufts University (2000).
[17] L.D. Bergman, J.S. Richardson, D.C. Richardson, and F.P. Brooks, "VIEW - An Exploratory Molecular Visualization System with User-Definable Interaction Sequences," Proc. ACM SIGGRAPH'93 Conference, pp. 117-126, Addison-Wesley/ACM Press, 1993.
[18] M. Conway, R. Pausch, R. Gossweiler, and T. Burnette, "Alice: A Rapid Prototyping System for Building Virtual Environments," Adjunct Proceedings of ACM CHI'94 Human Factors in Computing Systems Conference, vol. 2, pp. 295-296, 1994.
[19] M. Gleicher, "A Graphics Toolkit Based on Differential Constraints," Proc. ACM UIST'93 Symposium on User Interface Software and Technology, pp. 109-120, Addison-Wesley/ACM Press, Atlanta, Ga., 1993.
[20] M. Green and R.J.K. Jacob, "Software Architectures and Metaphors for Non-WIMP User Interfaces," Computer Graphics, vol. 25, no. 3, pp. 229-235, July 1991.
[21] M. Kass, "CONDOR: Constraint-Based Dataflow," Proc. ACM SIGGRAPH'92 Conference, pp. 321-330, Addison-Wesley/ACM Press, 1992.
[22] M.P. Stevens, R.C. Zeleznik, and J.F. Hughes, "An Architecture for an Extensible 3D Interface Toolkit," Proc. ACM UIST'94 Symposium on User Interface Software and Technology, pp. 59-67, Addison-Wesley/ACM Press, Marina del Rey, Calif., 1994.
[23] MPI Message Passing Interface Forum. "MPI: A Message-Passing Interface Standard". International Journal of Supercomputer Applications and High Performance Computing, 8(3/4), 1994.
[24] Peter S. Pacheco, Parallel "Programming with MPI", Morgan Kaufman Publishers, Inc. 1997.
[25] R.D. Hill, "The Rendezvous Constraint Maintenance System," Proc. ACM UIST'93 Symposium on User Interface Software and Technology, pp. 225-234, Atlanta, Ga., 1993.
[26] R.D. Hill, T. Brinck, S.L. Rohall, J.F. Patterson, and W. Wilner, "The Rendezvous Architecture and Language for Constructing Multiuser Applications," ACM Transactions on Computer-Human Interaction, vol. 1, no. 2, pp. 81-125, June 1994.
[27] R.J.K. Jacob, "A State Transition Diagram Language for Visual Programming," IEEE Computer, vol. 18, no. 8, pp. 51-59, 1985.
[28] R.J.K. Jacob, "A Visual Language for Non-WIMP User Interfaces," Proc. IEEE Symposium on Visual Languages, pp. 231-238, IEEE Computer Society Press, 1996.
[29] R.J.K. Jacob, "Eye Movement-Based Human-Computer Interaction Techniques: Toward Non-Command Interfaces," in Advances in Human-Computer Interaction, Vol. 4, ed. by H.R. Hartson and D. Hix, pp. 151-190, Ablex Publishing Co., Norwood,N.J.,1993.
[30] R.J.K. Jacob, "Using Formal Specifications in the Design of a Human-Computer Interface," Communications of the ACM, vol. 26, no. 4, pp. 259-264, 1983. Also reprinted in Software Specification Techniques, ed. N. Gehani and A.D. McGettrick, Addison-Wesley, Reading, Mass, 1986, pp. 209-222.
[31] R.J.K. Jacob, L. Deligiannidis, and S. Morrison, "A Software Model and Specification Language for Non-WIMP User Interfaces" ACM Transactions on Computer-Human Interaction, Vol. 6(1) pp. 1-46 (March 1999).
[32] S. Hudson and I. Smith, "Practical System for Compiling One-Way Constraint into C++ Objects," Technical Report, Georgia Tech Graphics, Visualization, and Usability Center, 1994.
[33] S.E. Hudson, "Graphical Specification of Flexible User Interface Displays," Proc. ACM UIST'89 Symposium on User Interface Software and Technology, pp. 105-114, Addison-Wesley/ACM Press, Williamsburg, Va., 1989.
[34] S.E. Hudson, "Incremental Attribute Evaluation: A Flexible Algorithm for Lazy Update," ACM Transactions on Programming Languages and Systems, vol. 13, no. 3, pp. 315-341, July 1991.
[35] S.K. Feiner and C.M. Beshers, "Worlds within Worlds: Metaphors for Exploring n-Dimensional Virtual Worlds," Proc. ACM UIST'90 Symposium on User Interface Software and Technology, pp. 76-83, Addison-Wesley/ACM Press, Snowbird, Utah, 1990.
[36] S.L. Tanimoto, "VIVA: A Visual Language for Image Processing," Journal of Visual Languages and Computing, vol. 1, no. 2, pp. 127-139, June 1990.
[37] Selic, Bran and Gullekson, Garth and Ward, Paul T. "Real Time Object Oriented Modeling". John Wiley & Sons, Inc., 1994.
[38] V. S. Sunderam, "PVM: A Framework for Parallel Distributed Computing, Concurrency: Practice and Experience", 2, 4, pp 315--339, December, 1990.
[39] Sandeep Singhal and Michael Zyda, "Networked Virtual Environments"Design and Implementation, Addison-Wesley, ACM Press, New York NY 1999.