Studying the Benefits and Challenges of Immersive Dataflow Programming

Lei Zhang
School of Information
The University of Michigan
Ann Arbor, MI USA
raynez@umich.edu

Steve Oney
School of Information
The University of Michigan
Ann Arbor, MI USA
soney@umich.edu

***Abstract*—Creating Virtual Reality (VR) applications normally requires advanced knowledge of imperative programming, 3D modeling, reactive programming, and geometry. Immersive authoring tools propose to reduce the learning curve of VR programming by allowing users to create VR content while immersed in VR. Immersive authoring can take advantage of many of the features that make VR applications intuitive and natural to use—users can manipulate programming primitives through direct manipulation, immediately see the output of their code, and use their innate spatial reasoning capabilities when viewing a program. In this paper, we investigate the benefits and challenges of immersive dataflow authoring. We implemented an immersive authoring tool that enables dataflow programming in VR and conducted a series of retrospective interviews. We also describe design implications for future immersive authoring tools.**

***Index Terms*—dataflow, immersive authoring, virtual reality**

I. Introduction

Virtual Reality (VR) applications can enable more natural human-computer interactions by matching the computer's representation of an environment with the spatial processing capabilities that humans have evolved over thousands of years [1]. By giving users a sense of presence and immersion, VR applications can nearly eliminate the gulfs of execution (how users translate intent into action) and evaluation (how users understand the state of a system) [2]. Although using VR applications can be natural and intuitive, creating VR applications requires specialized knowledge including advanced knowledge of imperative programming languages, three-dimensional (3D) modeling, reactive programming, and geometry.

One possible solution to the challenges of authoring VR content is to create Visual Programming Langauges (VPLs) for immersive 3D environments—to allow programmers to create content directly while immersed in VR. This paradigm is called *immersive authoring* [3], [4]. Immersive authoring tools have several potential advantages over traditional tools for creating VR content. First, immersive authoring environments can be intuitive, as they allow users to manipulate programming primitives through direct manipulation [5]—reducing the gulf of execution. Second, immersive authoring environments allow users to evaluate their code as they write it in the VR environment [4]—reducing the gulf of evaluation. Finally, by situating programs in an easily navigable 3D world, immersive authoring tools can leverage our natural spatial reasoning capabilities [1].

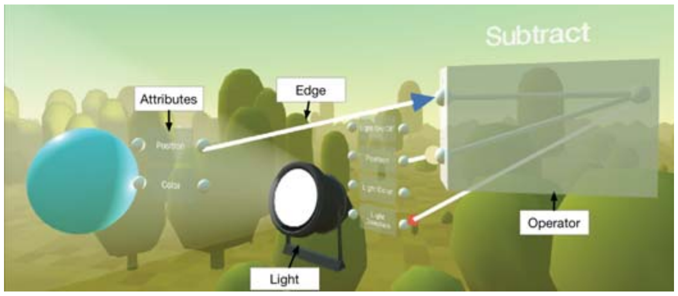


Fig. 1: A screenshot of our immersive dataflow programming tool. The directed arrows specify edges that determine the direction of data propagation. Operators accept any nunber of inputs and produce one or more outputs. The operator shown subtracts the position of the light from the position of the sphere, producing a direction from the light to the sphere. The result of the operator goes to the *direction* attribute of the light. This program thus creates a scene where the light always shoots at the sphere, even as the sphere and light move.

In this paper, we evaluate the challenges and benefits of immersive dataflow programming tools. Specifically, we created and evaluated an immersive dataflow programming language[1]. The results of our evaluation provide design insights that have implications for future immersive authoring tools.

II. Related Work

Our study builds on prior work in immersive authoring tools, 3D programming environments, and dataflow programming.

A. Immersive Authoring Tools

Immersive authoring allows users to create dynamic virtual scenes while immersed in a virtual environment. One of the earliest attempts to achieve this was Steed et al.'s dataflow representation for customizing behaviors [6]. Researchers have since built several immersive authoring systems, including iaTAR for creating AR scenes [3], [7], Ivy [8] for programming IoT devices, and Soundstage for creating music [9]. Each of these systems uses dataflow to represent behaviors. However, none of this prior work has studied the usability of dataflow in their immersive authoring tool, which is the

[1]Our immersive authoring tool is open source and publicly available: http://raynezhang.me/files/ImmersiveAuthoring.zip

focus of this paper. Of prior immersive authoring systems, only two (Ivy and Soundstage) run on modern VR hardware and only one (Soundstage) is publicly available. However, Soundstage was designed for authoring music. Thus, we built a new VR immersive authoring tool to use in our evaluation. However, our findings are generalizable to other immersive dataflow authoring systems, which use similar paradigms and interactions.

B. 3D Programming Environments

Game engines based on the entity-component architecture such as Unity [10] and Unreal [11] have been the most popular tools for programming 3D interactive applications. In recent years, the advances of WebVR have also given rise to libraries and frameworks such as Three.js [12] and A-FRAME [13], which enable developers to build VR scenes as web applications. However all these tools require expertise in imperative programming languages and do not allow users to prototype scenes while immersed.

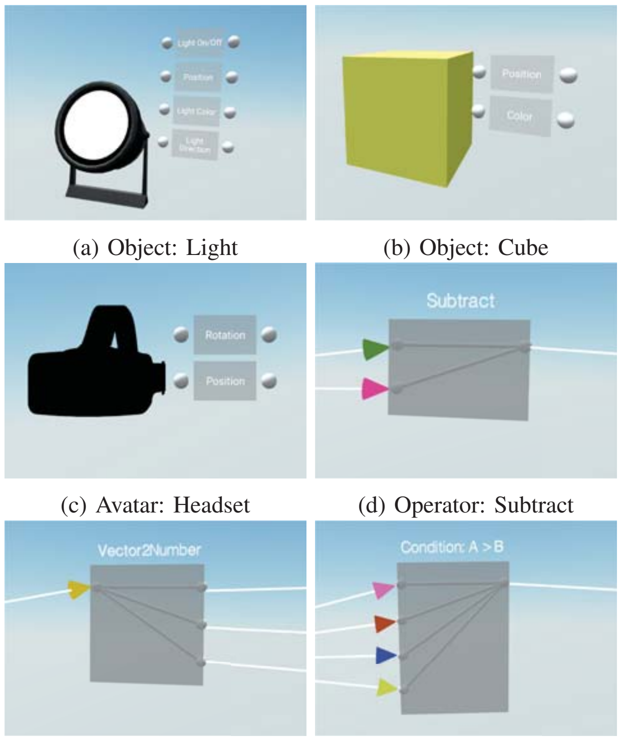
C. Dataflow Programming Languages

Dataflow programming languages have a long history, beginning with Bert Sutherland's Ph.D. thesis [14]. The dataflow model is represented by a directed graph, consisting of data sources, data sinks and nodes. The nodes are program objects or primitive operations. The direction of each edge represents the direction of the data propagation across different nodes. The dataflow programming paradigm has been the basis of many visual programming languages and used by successful commercial software such as LabView [15] and Max [16]. While being easy to operate and understand, dataflow programming languages also have several open problems [17], [18]. For example, complex dataflow programs can be visually cluttered and difficult to interpret. Further, few tools exist to help users effectively debug dataflow programs.

III. Immersive Authoring System

In our immersive authoring system, each operator is represented as a translucent box that takes inputs and produces outputs (Fig. 2d, 2e, 2f). Each input has a connector on the left side of the box and each output has a connector on the right side. For example, the *Subtract* operator (Fig. 2d) has two connectors on the left: + (plus) and - (minus), where data inputted through the + connector will be added to the result and data inputted through the - connector will be subtracted from the result. The final result will be propagated through the output connector on the right.

Users can also create behaviors that depend on the position of their headset and controllers through *avatars*, which are proxies of their headset and controllers (Fig. 2c). Avatars contain output nodes for the position, rotation, and each button on these devices. Each avatar can be viewed as a node and by drawing edges between the avatar and other virtual object in the scene, users can create interactive scenes where the attributes of the virtual objects will depend on the the user's tracked devices (i.e. the headset and the controllers).



(a) Object: Light (b) Object: Cube

(c) Avatar: Headset (d) Operator: Subtract

(e) Operator: Vector2Number (f) Operator: Condition $A > B$

Fig. 2: Examples of objects, avatars, and operators. Objects (a&b) have several attributes listed next to them that can be modified. Avatars (c) represent virtual proxies of users' inputs. Operators (d, e, f) are computational units that take inputs produce the results as outputs.

Users use two controllers to interact with the immersive authoring tool. Users can create objects, avatars, and operators through a palette tool menu, which is "attached" to the user's left hand controller and whose items can be selected by pointing (via raycast) and pressing a trigger on the right hand controller. The application employs a drag-and-drop interaction for drawing edges using the raycast and provides straight arrows as intermediate feedback (Fig. 3 C).

IV. Study

To better understand the benefits and challenges of immersive dataflow programming, we conducted a user study with our immersive authoring tool. Although this user study was only conducted with our tool, we believe our findings are representative of other immersive authoring systems [3], [6]–[9], which also employ similar dataflow metaphor, visualizations, and node placement features in VR.

We recruited 7 participants (2 male, 4 female, and 1 prefer not to say), ages 20–27 ($\mu = 24.1$), from the authors' university. All participants had at least basic programming experience (having completed at least one programming class). Two participants also had experience creating VR applications. We compensated every participant with $25 in cash for their participation. Our application was run on the Oculus Rift on a Windows 10 system with an Nvidia GTX 1080 GPU, Core i7 CPU and 16 GB RAM.

A. Procedure

Every study lasted 90 minutes. Participants spent the first 30 minutes in a tutorial that walked them through a series of small tasks including creating and manipulating objects, drawing edges between nodes, and using different operators. The tutorial also gave participants a chance to ask questions and experiment on their own.

We then asked participants to perform two tasks:

- *Task 1*: Create three spotlights and make them shoot at and follow the user. The three spot lights should emit light of random colors.
- *Task 2*: Create a scene where a spot light will shoot at a cube when the user's left hand is higher than the user's right hand, and the spot light will shoot at a sphere when the user's left hand is lower than the user's right hand.

We gave participants 15 minutes for each task and did not give them further instructions on how to complete the tasks unless they specifically requested help. We then conducted a one-on-one retrospective interview with each participant.

B. Results

Most participants expressed that the application is fun to play with:

> P3: *"I really like to play with (it). I think it's really good to explore. It was just really fun."*

Participants also generally enjoyed the immersive feeling of being able to place things freely and naturally in the 3D virtual space as if they were manipulating them in the real world:

> P2: *"I really enjoyed having stuff in a three-dimensional space... it just feels so real. "*

When being asked about any confusion about the application, all participants commented that drawing edges between nodes is difficult and annoying since they often missed the target connector when it became too small and hard to aim at in the scene:

> P1: *"The aiming of putting a line on a circle was annoying. I messed up five times or something..."*

Another difficulty from most participants is the struggle to follow the execution of the program when it gets more complicated and the lines get cluttered. Based on that, some participants expressed desire for adding secondary notations (e.g. comments, annotations, etc.) or grouping nodes into sub programs:

> P4: *"I think it will be helpful to let the user to group things together. For example, for all the condition patch(es), I can group them together and have a note like 'this is gonna compare positions' "*

When being asked to compare the application with their previous experiences in text-based programming, most participants commented positively on this application and all participants expressed that the application is easier for beginners and is therefore suitable for educational use:

> P2: *"This tool reminds me of this thing called Alice. I can definitely see it being a potential educational tool. I think it can make things so much easier."*

V. Discussion

In this section, we discuss several challenges and benefits of immersive dataflow programming based on the aggregated results above. For each challenge, we provide our insights and design implications based the study.

1) Layout Management: From the results, users were not strategic about where they placed the nodes in their dataflow programs. However, they typically placed related nodes next to each other, either horizontally or vertically. They normally placed nodes in a from-left-to-right order based on the direction of the data propagation. At the same time, users indicated that lines became clustered and hard to follow when the program became more complicated. This is also a challenge in general dataflow programming languages but a compounding problem was that edges could be occluded and hidden because of the wealth of depth information. One design implication for managing the dataflow program layout is to automatically sort and place the nodes and edges, according to some participants. Another implication is to support objects customization for the users. Specifically, participants commented that being able to group related nodes and edges together is helpful for keeping track of the dataflow program. They also expressed that being able to add annotations or comments would be helpful for understanding each part of the dataflow program.

2) Drawing Edges: During the retrospective interviews, most participants preferred drawing edges through direct manipulation (Fig. 3 A), as opposed to using raycast to aim at target connectors (Fig. 3 C). There are two reasons for this. One reason is that through direct manipulation users can feel more immersed—as if the wire is in their hands. The other reason is that it is easier to aim and connect when the connector is close to the users. This is a challenge that is specific to 3D immersive environments since users draw edges in a 3D space using the controllers held in their hands as opposed using the two-dimensional (2D) Window Icon Menus Pointer (WIMP) interfaces.

We therefore propose four mechanisms for drawing edges, as shown in Fig. 3. Each mechanism has its own tradeoffs. Users can draw edges through direct manipulation as if they are holding the wires. The first method (Fig. 3 A) allows users to draw edges through direct manipulation as if they are holding the wires. One benefit of this method is that it is intuitive by allowing users to connect edges in the same way that they do in the physical environment. Another benefit is that it allows customized shapes of the edges, which is helpful in avoiding occluding edges with each other. The drawback of this method is that it is hard to draw an edge between two objects that are far away from each other without moving in the virtual world. The second method (Fig. 3 B) allows users to draw edges using a proxy at the fixed distance to the controller. This method offers custom edges but also requires them to move around the scene to connect

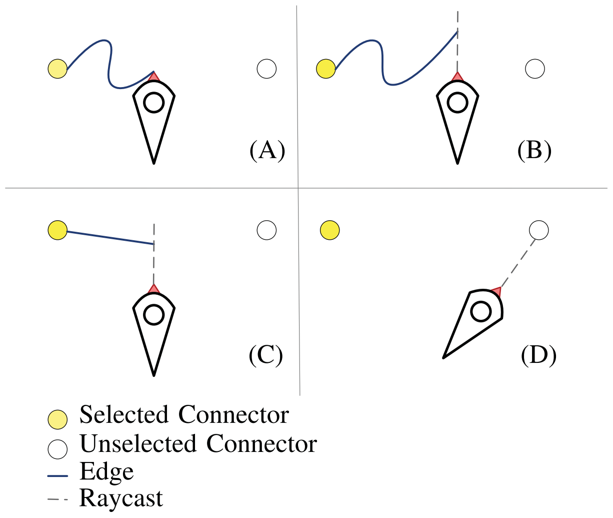


Fig. 3: An illustration of four alternative techniques for specifying edges between nodes in VR using a controller. Edges can either be drawn as custom shapes (A&B) or as a straight line between nodes (C&D). Users could draw edges directly with their controller (A); along two dimensions with a depth that the system computes (B); as straight edges between nodes while showing intermediate feedback (C); or by selecting source and target nodes with no intermediate feedback (D).

distant nodes. The third method (Fig. 3 C) allows users to draw straight edges using the raycast and shows the straight edge as intermediate feedback. The benefit of this approach is allowing users to draw lines beyond their reach without moving in the virtual world. However, it is hard to aim at the connector that is too far away and appears very small in the scene. The last method (Fig. 3 D) is similar to mouse-clicking, where users will click at the first connector and then the second connector in order to draw an edge. This method produces no intermediate feedback and avoids the drag-and-drop interaction. However, the drawback of this approach is the lack of direct manipulation.

3) Navigation: Changing the user's viewpoint of the dataflow program is also challenging when the whole program is embedded in an immersive 3D space. This is also different from dataflow programming languages on 2D WIMP interfaces where users can pan and zoom with the mouse. Instead, in the immersive virtual environments, users have to move around in the virtual world in order to navigate through the program. This usually causes the feeling of lack of control of their own bodies when the locomotion of self affects the viewpoint of the dataflow program. To cope with this challenge, we propose a 2D map-like dataflow representation design, where users can zoom in, zoom out, and move the whole diagram without changing their positions in the world.

4) Immersion and Natural Interaction: Immersive dataflow programming tools allow users to manipulate and place things freely in the 3D space in the same way that they interact with the physical world. Multiple users expressed that being able to create and place objects (such as cubes and lights) wherever they want is the most enjoyable part of the immersive dataflow programming tool. Participants' sense of immersion was also enhanced by the immediate feedback they received. Users can see the changes instantly after drawing an edge between two nodes, which makes it faster to prototype VR scenes and faster to make changes to the existing program.

5) Potential for Educational Use: Most participants agree that this tool is easier for beginners and has the potential for educational use. This may be due to that existing imperative programming tools have steeper learning curve than immersive dataflow programming tools. Some participants also expressed that they found the immersive tool to be fun and engaging to interact with.

Based on the results of our study, we do not see an advantage in allowing users to place dataflow objects themselves in 3D space. Users in our study did not tend to spend the time to organize their dataflow nodes and occlusion could make it difficult for them to follow the edges. However, we do see advantages in including dataflow languages inside of immersive authoring environments. Participants specifically liked the quick feedback loop and the ability to connect dataflow outputs and inputs directly to objects in their environment and objects that represented the user.

Thus, we feel that dataflow is still an effective option for immersive authoring. However, there is little benefit to giving users complete 3D freedom in dataflow authoring. Instead, we propose dataflow interactions that happen in 2D—where all of the nodes and operators are visible on a 2D plane but can be connected to objects in the 3D space. For example, one could imagine a "breadboard" metaphor where users can see their dataflow program on a 2D plane but they can connect the output of their dataflow diagrams to objects in the virtual world.

VI. Limitations & Conclusion

There are several limitations of the presented study. First, we only recruited participants who were willing and able to use VR, which may bias the kinds of feedback that participants give. Second, we performed a short-term study and received only the first impression of the immersive dataflow programming tool on the participants. Despite having the training session, some participants expressed that it was difficult to get familiar with all the features in a short time. A longitudinal study would be necessary to better understand participants' learning curve.

In this paper we presented an exploratory study analyzing the benefits and challenges of immersive dataflow programming tools. We also proposed several design implications based on the results. We believe that dataflow programming is an effective approach for immersive authoring and that there is ample room for future improvement to address the design challenges brought by the freedom of 3D space.

VII. Acknowledgements

We thank Hariharan Subramonyam, Michael Nebeling, Minje Choi, Jane Im, and Zhuofeng Wu for their feedback on the immersive authoring application. We also thank our study participants and reviewers for their time and effort.

References

[1] R. S. Kalawsky, *The science of virtual reality and virtual environments: a technical, scientific and engineering reference on virtual environments*. Addison-wesley Workingham, 1993.

[2] D. A. Norman and S. W. Draper, *User centered system design: New perspectives on human-computer interaction*. CRC Press, 1986.

[3] G. A. Lee, C. Nelles, M. Billinghurst, and G. J. Kim, "Immersive authoring of tangible augmented reality applications," in *Proceedings of the 3rd IEEE/ACM international Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2004, pp. 172–181.

[4] G. A. Lee, G. J. Kim, and M. Billinghurst, "Immersive authoring: What you experience is what you get (wyxiwyg)," *Communications of the ACM*, vol. 48, no. 7, pp. 76–81, 2005.

[5] B. Shneiderman, "Direct manipulation: A step beyond programming languages," in *ACM SIGSOC Bulletin*, vol. 13, no. 2-3. ACM, 1981, p. 143.

[6] A. Steed and M. Slater, "A dataflow representation for defining behaviours within virtual environments," in *Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium*. IEEE, 1996, pp. 163–167.

[7] G. A. Lee and G. J. Kim, "Immersive authoring of tangible augmented reality content: A user study," *Journal of Visual Languages & Computing*, vol. 20, no. 2, pp. 61–79, 2009.

[8] B. Ens, F. Anderson, T. Grossman, M. Annett, P. Irani, and G. Fitzmaurice, "Ivy: Exploring spatially situated visual programming for authoring and understanding intelligent environments," in *Proceedings of the 43rd Graphics Interface Conference*. Canadian Human-Computer Communications Society, 2017, pp. 156–162.

[9] Soundstage vr. [Online]. Available: https://github.com/googlearchive/soundstagevr/

[10] Unity. [Online]. Available: https://unity.com/

[11] Unreal. [Online]. Available: https://www.unrealengine.com/

[12] three.js. [Online]. Available: https://threejs.org/

[13] A-frame. [Online]. Available: https://aframe.io/

[14] W. R. Sutherland, "The on-line graphical specification of computer procedures." Ph.D. dissertation, Massachusetts Institute of Technology, 1966.

[15] Labview. [Online]. Available: http://www.ni.com/labview/

[16] Max. [Online]. Available: https://cycling74.com/products/max/

[17] W. M. Johnston, J. Hanna, and R. J. Millar, "Advances in dataflow programming languages," *ACM computing surveys (CSUR)*, vol. 36, no. 1, pp. 1–34, 2004.

[18] T. B. Sousa, "Dataflow programming concept, languages and applications," in *Doctoral Symposium on Informatics Engineering*, vol. 130, 2012.