PoseVEC: Authoring Adaptive Pose-aware Effects using Visual Programming and Demonstrations

Yongqi Zhang
George Mason University
USA
yzhang59@gmu.edu

Cuong Nguyen
Adobe Research
USA
cunguyen@adobe.com

Rubaiat Habib Kazi
Adobe Research
USA
rhabib@adobe.com

Lap-Fai Yu
George Mason University
USA
craigyu@gmu.edu

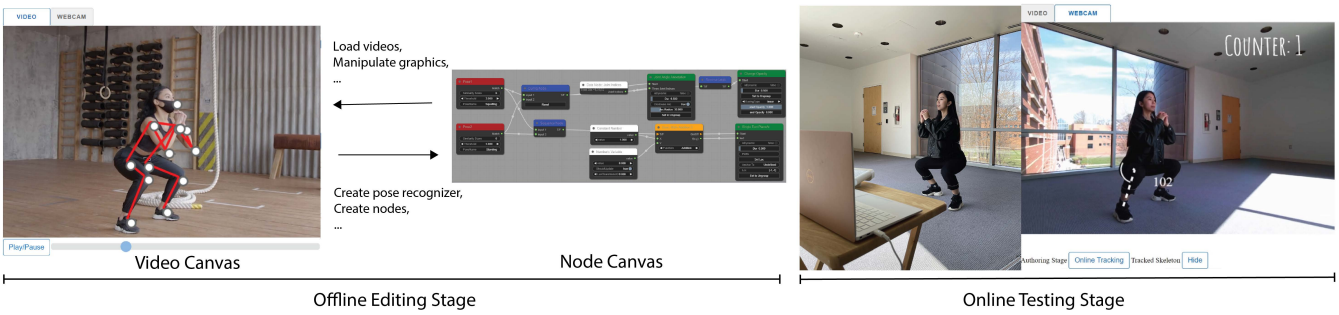

Figure 1: PoseVEC has two authoring stages for creating pose-aware visual effects: offline editing and online testing. Upon loading an input video, PoseVEC will switch to the offline editing stage. PoseVEC also supports editing with video demonstration, users can directly interact with the video canvas to add pose-related configuration to the node program. For example, Users can scrub the video timeline to create a squatting-down pose recognizer(red node in node canvas). When users finish with their node program, they can switch to the online testing mode to see the pose effect from another video or in front of a live webcam. ©Pexels

ABSTRACT

Pose-aware visual effects where graphics assets and animations are rendered reactively to the human pose have become increasingly popular, appearing on mobile devices, the web, or even head-mounted displays like AR glasses. Yet, creating such effects still remains difficult for novices. In a traditional video editing workflow, a creator could utilize keyframes to create expressive but non-adaptive results which cannot be reused for other videos. Alternatively, programming-based approaches allow users to develop interactive effects, but are cumbersome for users to quickly express their creative intents. In this work, we propose a lightweight visual programming workflow for authoring adaptive and expressive pose effects. By combining a programming by demonstration paradigm with visual programming, we simplify three key tasks in the authoring process: creating pose triggers, designing animation parameters, and rendering. We evaluated our system with a qualitative user study and a replicated example study, finding that all participants can create effects efficiently.

CCS CONCEPTS

• **Human-centered computing** → *Systems and tools for interaction design*; *User centered design.*

KEYWORDS

Visual effects authoring, motion graphics, pose recognition, Programming by Demonstration

ACM Reference Format:
Yongqi Zhang, Cuong Nguyen, Rubaiat Habib Kazi, and Lap-Fai Yu. 2023. PoseVEC: Authoring Adaptive Pose-aware Effects using Visual Programming and Demonstrations. In *The 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23), October 29–November 01, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3586183.3606788

UIST '23, October 29–November 01, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0132-0/23/10.
https://doi.org/10.1145/3586183.3606788

1 INTRODUCTION

Human-aware motion graphics refer to graphical assets and animations that can be rendered reactively to changes coming from tracked biometric data such as face, hand, or body pose. These types

Figure 2: Examples of pose-aware visual effects. Left: A dance matching effect and a counter effect Instagram filter that show the number of dance poses the user could replicate successfully[29]. ©Petricore Right: A stylized text-shaking animation for a marketing campaign[27]. ©Shotopop

of content are growing in popularity in many creative domains like visual effects in films or interactive augmented reality (AR) filters in mobile applications such as Snap, Instagram, or Tiktok. In particular, *pose-aware visual effects*—motion graphics designed to be triggered or controlled by a user's body pose has gained a lot of attention in sport tutorials, AR lens and video effects creation. Figure 2 shows some examples of pose-aware visual effects, which we henceforth refer to as *pose effects*. However, creating adaptive and expressive pose effects still remains a challenging task for visual effect designers. Traditionally, these effects are often produced by using keyframe-based approaches [4, 19**?**] where animations are baked into the video timeline. As a result, the output pose effects are non-adaptive, which means the designers cannot reuse a pose effect on another video or on a live video stream. For instance, when creating a pose effect similar to the one shown in Figure 2, if the person in the video slightly turns around, or changes to a different pose, then the effect designer will have to adjust the existing animations to adapt to the new changes in poses. This cumbersome process makes it difficult for the designer to reuse an existing effect or adapt it to a new creative idea.

In recent years, various dedicated tools such as Lens Studio, Spark AR, and Tiktok Effect House have been created to promote developing pose effects *programmatically*, with the aim of making the authoring workflow both expressive and adaptive. This approach gives designers a lot of flexibility to fully customize the low-level details such as how a pose is tracked and how pose data is transformed and mapped into the video animation. This flexibility empowers users to create more expressive effects. More importantly, the created effect is now an executable program that could be used on another input video. This "reusability" benefit is significant because it promotes community sharing and experimentation.

However, programming-based workflow is notoriously difficult for non-technical designers. In order to create a pose effect, designers typically need to accomplish three key technical work. Figure 3 provides an overview of these tasks. *1) Pose recognition*: they first need to set up a pose recognizer from the input videos, which often

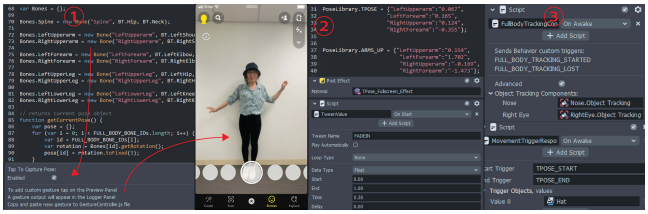

Figure 3: Current workflow using Lens Studio to create a pose effect: 1) pose recognition: obtain T-pose joint data using a provided function and load a T-pose video into the system to acquire T-pose data; 2) animation design: storing and transforming the pose data into animation parameters using another script; 3) rendering: Using a body tracking tracker to associate a virtual effect with body parts, and modifying the movement trigger response script to set up the trigger response for the T-pose effect.

requires them to manually write various functions to obtain joint configurations, setting up an algorithm to perform pose matching, and fine tune the detection threshold. *2) Animation design*: in this step, users transform low-level pose data like joint index, distance, or angle into animation parameters and establish some parameter mappings between these data and the graphics that they want to animate. *3) Rendering*: finally, they need to establish a render loop for that can render graphical assets based on the animation parameterizations in step two and the pose recognition events in step one. All of these steps are tedious and time-consuming as they require significant technical expertise in handling low-level pose data and writing event-driven animation codes.

In this paper, we investigate a visual creative workflow for authoring adaptive and expressive pose effects. Our approach leverages both Programming by demonstration (PbD) and visual programming paradigms to simplify the programming-based workflow for non-technical designers. Specifically, we allow a user to interact with both a video canvas and a node programming canvas to craft a pose effect. For node programming, we adapt the dataflow programming model [35, 38] to the task of crafting interactive pose effects. Our model consists of five types of nodes to assist in pose recognition (e.g., pose recognizer node & logic node), animation design (data node & transform node) and rendering (render node). Users can connect one node to another to craft an effect and see it rendered on the video canvas in real-time. In addition, users can directly interact with the video canvas to quickly specify complex pose configurations such as pose skeleton data, joint data, or pose-relative asset positioning. This unique combination allows us to use the video canvas as both an output panel and a companion editor, simplifying the visual programming workflow even further.

To this end, we developed a proof-of-concept web authoring application called PoseVEC (**Pose**-aware **V**isual **E**ffect **C**reator). Our goal is to use PoseVEC to examine how users could use both PbD and visual programming in crafting expressive and adaptive pose effects. PoseVEC supports both video from a live webcam input or from an uploaded file. To enable adaptive behaviors in pose effects, we leverage a one-shot pose embedding machine learning (ML) network [34] to perform real-time pose matching on the source video. This core detection mechanism is represented as a node

in PoseVEC. More importantly, users could use other nodes in PoseVEC to iteratively add animation behaviors on top of the initial pose detection node. This process allows a user to use PoseVEC to quickly create pose effects that are both expressive and adaptive. The resulting effects can be reused on a different video or in a live video with minimal or no configuration needed. In a sense, PoseVEC empowers users to craft reusable *template* effects rather than baked-in effects. It could enable novel visual effect applications. For example, video editors could save editing time by developing a pose effect once and apply it to new videos or to existing videos with compatible poses. Designers could use PoseVEC to mock up new AR lens effect by designing on an upload video and testing on the the live webcam view.

We performed two evaluations to assess the effectiveness of our tool: a first-use study to gather feedback on our authoring workflow and the usability of PoseVEC, and a replicated example study to demonstrate the utility and expressiveness of PoseVEC. In summary, our contributions include:

- The design of a new approach that combines PbD and visual programming to help users author adaptive & expressive pose effects using human poses.
- The instantiation of this workflow in a proof-of-concept web authoring system called PoseVEC. PoseVEC provides users with a node graph model that was designed specifically to ease pose effect authoring. It also integrates direct video interaction to further simplify and speed up user workflow.
- A qualitative evaluation and a replicated examples demonstration to examine the usability and utility of PoseVEC.

2 RELATED WORK

2.1 Pose Effects Applications and Authoring

Authoring poses effect refers to the task of adding visual effects to a video that involves human subjects. These visual effects can either be directly applied to human subjects(e.g., contouring the body part) or be driven by human actions to create the illusion that the assets are part of the video and respond to the pose movement. There are two main approaches to author pose effects:post-production and the programming.

The post-production approach is more prevalent in filmmaking, where video editors use tools such as After Effects to add keyframes for graphical assets and animation to the video timeline. Despite the rich features available in these software to help users in motion graphics creation and editing, it may be too complex for amateurs to learn to use. To address this challenge, researchers have attempted to streamline the motion graphics creation process for amateurs. For example, Katika [13], is an end-to-end tool for authoring motion graphics videos that aim to help users understand the process of motion graphic creation and create motion graphic videos without external guidance. Similarly, PoseTween [19] is a system designed for novice users to create virtual object animations by leveraging human motion. In contrast, our work focuses more on the programming approach. In particular, we investigate how to simplify the process of authoring pose effects using programming workflow. Compared to the post-production approach, developing effects programmatically allows the creator to have more creative freedom.

Many complex and novel applications of interactive body-based experience using programming workflow have been explored. For instance, Anderson et al. has built a Mirror-based augmented reality system to record and learn physical movement sequences [6] This system uses the body-tracking technique and skeleton comparison to provide posture guidance and feedback in real-time. Another example is PoseBlocks [14], a block-based programming toolkit designed for educating students on building interactive physical movement-based experiences. This toolkit incorporates AI-powered face, hand, body tracking technique into blocks for students. RealitySketch [36] leverages interactive sketching to parameterize real world objects and create interactive in-situ AR visualizations driven by those parameters.

Moreover, some research focuses on virtual puppetry techniques to create animations for non-human objects, where virtual objects are manipulated through physical control such as keyboard control and body-tracking sensors. For instance, Chen et al propose an interactive system that allows novice users to scan and animate real-world objects by employing a deformation method to process skeleton information and object geometry [8]. Similarly, Seol et al. proposed a real-time motion puppetry system that allows users to manipulate the motion of non-living creatures naturally through direct feature mapping and motion coupling [32] In all of these examples, the high barrier of entry associated with the programming-based approach is apparent, as it typically requires a team of researchers or engineers to write programs from scratch in order for users to create pose effects. Our research makes the first attempt into looking at how to simplify this workflow, such that normal designers could also directly learn and create pose effects.

2.2 Visual Programming for Interactive Animations

Visual programming is a common approach to reducing programming complexity, including declarative programming, dataflow programming, or block-based programming. These approaches typically present users with a visual abstraction in the form of blocks or nodes. Each of these nodes could represent a source of data or functions or operations to be executed. By setting up and connecting these nodes, users could construct a program in a declarative manner. Dataflow programming is most often used in helping users work with complex 3D rendering pipelines to produce shaders [33] or materials [3]. In contrast, our research focuses on helping users create interactive behaviors. To this end, numerous dataflow programming models have been developed for a wide variety of tasks such as gesture programming [15, 21], animations [28, 31], vector drawing [12], and VR programming [10, 38]. And yet, none of these works have focused on pose effects design. More recently, social media platforms like Tiktok and Snap have started introducing creator tools for users to create AR filters based on body tracking, allowing users to create and share interactive pose effects. However, their dataflow programming is not specific to pose effect design. In designing PoseVec, we identified three key tasks for pose effect programming and developed a set of nodes and authoring interactions to simplify these tasks. Briefly, these features support creating pose recognition, animation design, setting up the rendering loop, and refining and test. See Sec. 3 for more task details.

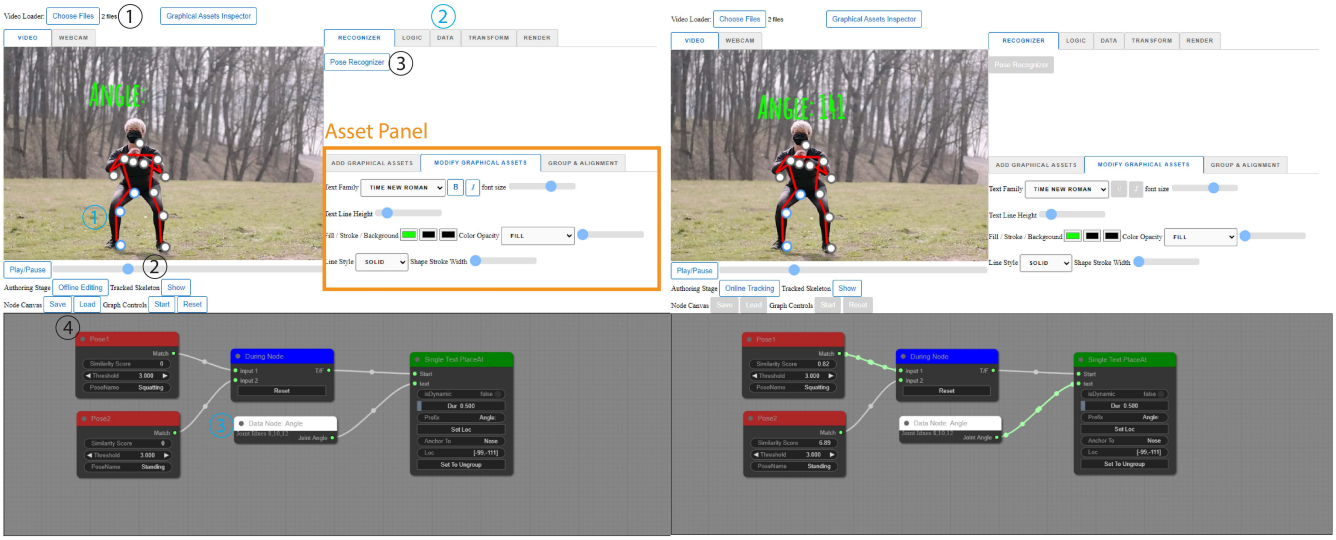

Figure 4: Overview of how a user could author and test a pose effect using PoseVEC. Here the user is creating an effect that could dynamically render the angle value of the right knee on the video. Left: The PoseVEC interface in the offline editing mode. Right: The PoseVEC interface in the online tracking mode. In the offline editing mode, the steps (in black numbered annotations) for creating a pose recognizer are as follows: (1) The user uploads a video to PoseVEC. (2) He can scrub the video timeline to select a pose of interest. (3) He presses "Pose Recognizer" button. (4) A pose recognizer node will add to the video canvas. Additionally, the steps (in blue numbered annotations) for creating a pose-related data node through joint selection on video canvas are shown: (1) The user selects the right hip, knee and ankle joint from the skeleton. (2) He presses the "Joint Angle Data" button under data tab. (3) A Joint Angle Data node is added to node program, which will output selected joint angle information when the node program runs. In the online tracking mode, the user can see the pose effect happening on the video canvas and observe the data flow in the pose program. ©Pexels

2.3 Integrating Visual Programming with Programming by Demonstration

One main issue of visual programming systems is that it can be slow for users to describe complex behaviors quickly. For example, when describing geometric configurations of a particular pose of interest, it is usually faster and easier for users to provide a system with some examples rather than constructing the example in a declarative manner. To this end, several recent works have started integrating programming by demonstration components into visual programming systems. For example, in GestureStudio [22], each gesture is recorded and visualized in a timeline block, which later can be assembled with other gestures or attached callback action. Similarly, Gesture Knitter [24] allows designers to provide hand gesture demonstrations first, then convert them into primitive blocks. This approach enables the creation and customization of complex gesture recognizers.

In addition, recent research utilized this new workflow approach in AR experience authoring. GestureAR [37], for example, enables freehand AR authoring and AR interactive experience. In the authoring process, users start by recording hand gestures, which are then associated with virtual content behaviors using a visual programming interface. Subsequently, they can test interactive content in real-time. Similarly, Rapido [18] focuses on video prototyping using AR-enabled mobile devices. Designers first draw sketches and demonstrate user intention in a video embedded with AR information. Then they use Rapido to convert the video prototype into an executable state machine, such that they can switch between these representations to test and refine the prototype. Teachable Reality [25] employs vision-based interactive machine learning to author tangible AR prototypes in-situ, by enabling the user to detect, train, bind, and author physical-virtual interactions.

Inspired by these works, we adopt a similar strategy in the new problem domain of pose effect authoring. Existing commercial workflows that adhere to the "No Inferencing" PbD model, as proposed by Myers et. al. [26].), often require users to complete tedious programming tasks (see Figure 3-1) in order to create pose effects. In contrast, we follow the "sophisticated AI Algorithms" PbD model and leverage a pre-trained one-shot pose embedding model [34] to accelerate the process of creating and fine-tuning a pose recognizer node. Another key idea in our PbD approach is to enable users to interact directly with the video canvas to provide complex pose configurations to the node program. Users could interact with the pose skeleton joint on the video to select a joint of interest, compute joint-wise parameters like distance and angle, or position assets relative to some joints. By identifying and offsetting these tasks to the video canvas, we create a more streamlined programming workflow for pose effect designers.

3 SYSTEM OVERVIEW

In this section, we outline the key components of our system and illustrate how these components are used to produce a pose effect. We follow a hypothetical user, Alice, as she designs a pose effect illustrated in Figure 4. We show how Alice could craft this pose effect using the visual workflow in PoseVEC without having to resort to using any explicit programming.

For this effect, Alice wants to display an overhead text that shows the joint angle of the right knee while the person is squatting down. The text is anchored to the person's head and moves along with the head movement during squatting. Please refer to the supplementary video for the complete design process and the final pose effect. Detailed definitions of the pose program and node designs are in Section 4. Alice starts by uploading a video of someone doing the similar squat pose that she found online.

Pose Recognition. The pose recognition trigger is the most important component in PoseVEC as it drives pose effect and animation to occur. After Alice uploads a video to the scene, she drags around the slider bar to select an interesting pose. Then, she presses the 'Pose recognizer' button under the recognizer tab to add pose recognizer to the node program. As her design goal is to create a pose effect for a squat-down video, she adds a standing pose and a squatting pose trigger to the node program. See Figure 4 (black numbered annotations) for an illustration of these steps. Please refer to Section 4.2 for more technical details of the pose recognition node.

Animation Design. Next, Alice needs to obtain joint angle of the right knee and then render this value on top of the person's head. Using PoseVEC, Alice needs to add two more nodes to the current node program: the Joint Angle Node from the Data Node category and the Single Text Render node from the Render Node category. First, she creates a Joint Angle Data node by selecting the right hip, knee, and ankle joints. This Joint Angle Data node will output the selected joint angle information as the node program runs. Then, she adds a Single Text Render node to the node program, which will automatically add text to the video canvas. She then drags the text to her head and holds it for one second to anchor the text to her head. We explain this anchoring operation in more details in the System Design section below (Figure 5). Lastly, she saves this anchoring information by clicking on the render node. To make the text more intuitive, she also inputs "Angle" as the prefix of the final text. Finally, she connects the Joint Angle Data node to the input of the Single Text Render node to complete the pose effect design. See Figure 4 (blue numbered annotations).

Rendering. Now, Alice needs a logical condition that controls the timing of the appearance of this pose effect so that it only appears while the person is squatting down. She adds a During node from the Logic node category. This During node takes two boolean inputs and constantly outputs a boolean value. It outputs true as long as input 1 is true before input 2 turns true. In this context, Alice uses it to ensure that the pose effect is triggered only when the transition of poses, from the squatting position to the standing position, is detected. She then connects the squatting down pose trigger to input 1 of during node and the standing pose trigger to input 2. Finally, she connects the output of during node to the input of the joint angle node for completing this node program. See Figure 4, bottom-left).

Refining and Testing. After Alice finishes all node connections, she switches the offline editing mode to online tracking mode. Immediately, she can see that the right knee joint angle is highlighted while the person is squatting down. This is because the similarity score in the squatting pose recognizer is below the threshold, leading to a true signal to trigger the animation. Alice can also observe how data flow from one node to another at some key moments. (See Figure 4, right)

Once Alice is satisfied with the pose effect, she can upload another video of a squatting pose or turn on the webcam to test the pose effect. Alternatively, she can upload a different sport video and refine the pose program slightly to adapt the same pose effect to a different sport pose. All she needs to do is to drag around the video slider and adding more pose recognizers to the current program.

4 SYSTEM DESIGN

We designed PoseVEC to help users craft expressive pose effects using visual programming. To make the system easy to use for non-technical designers, we have three main design goals:

(1) Visual workflow: providing a node programming UI that can support all key tasks in programming pose effects, including asset import, pose recognition, animation design, and rendering.
(2) PbD via video interaction: simplify the programming workflow by allowing users to directly interact with the input video stream to specify pose-related configurations for the node program.
(3) Ease of testing: providing visualizations and tools that allow users to quickly see results, test, and refine the pose effect program.

4.1 Programming a Pose Effect

Figure 4 shows an overview of PoseVec. A user can craft a pose effect by interacting with both the video canvas and the node canvas. A user can start by creating an input video source. We support both live webcam stream and offline video upload. Upon selecting an input video source, PoseVEC will switch to the editing mode.

Editing with Nodes. In the editing mode, a user can add graphical assets (Figure 4, top-left) to the video canvas and modify them. They can also add nodes to the node canvas to construct pose effects. Note that the node program remains inactive during this stage as there is no data flow in the connections. Our node program follows the dataflow programming model [38] where connected nodes form a direct graph and data can flow from the leftmost source nodes (e.g., pose recognizer & nodes) to the rightmost sink nodes (e.g., render nodes). Users can establish connections between nodes by dragging an edge from one node and connecting it to another node. While dragging the edge, input slots that have different input types will be disabled, allowing users to easily identify which input slots can be connected.

Editing with Video Demonstrations. During editing, users could interact with the video canvas to provide additional pose-related configurations to the node program. Table 1 provides an overview of which nodes could be created via the video canvas, these nodes are marked with "video-linked". Video-linked nodes can store pose-related data and can monitor and update its stored

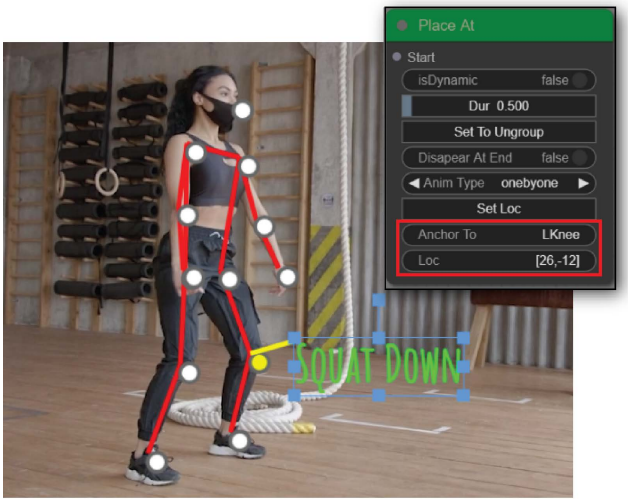

Figure 5: PoseVEC supports anchoring to the skeleton option. To do this, users can drag and hold an asset (e.g., the text "SQUAT DOWN") for one second, then PoseVEC will highlight the closet joint to which the asset can be anchored with a yellow line. This anchoring information could then be stored in the render node that controls the asset movement (e.g., MoveTo Node) so that the anchoring information can be applied during rendering. ©Pexels

data. For example, to create a pose recognizer node, users could scrub the video timeline to select the desirable pose and click on the pose recognizer node button on the UI. Likewise, selecting a joint on the pose and clicking on a Joint ID data node will create a data source for the corresponding joint. A user can also add, position, or anchor graphical assets to the pose to more easily define how an asset should be rendered relative to the selected pose. Selecting an asset and clicking on a render node will add it to the node canvas.

Testing Mode. To test the node program, a user can switch to the online tracking mode. In this mode, "source" nodes like recognizer and data will start propagating data to the rest of the node graph. For example, the pose recognizer node will leverage a pose similarity ML model to output a true trigger event if the stored pose is similar to the current pose in the video window. The data node will simply output the data that was set when the node is created. Eventually, when the data reaches the "sink" render nodes, the assets on the video canvas are rendered accordingly, creating the resulting animations of the pose effect. Running the node program in this way gives users flexible options to test the pose effect. They could see the effect on a static frame, a playback of a video file, or a live webcam stream. Note that in this mode, users cannot modify graphical assets or nodes, they can only observe the behavior of the node program and switch back to the editing mode to refine the pose effect. When the node program is executed, we render line animations to visualize how the data moves through the graph to help users better understand the model. See Figure 4 (bottom-right).

4.2 Node Types

Now we will describe in more details the design of PoseVEC's node model. A complete description of our node model is included in the Appendix. Overall, we have five main types of nodes. These nodes were designed specifically to support three main tasks in authoring pose effects: pose recognition (pose recognizer node & logic node), animation design (data node & transform node), and rendering (render node).

4.2.1 Recognizer Node. When added to the node canvas, PoseVEC runs a one-shot pose embedding network to transform the currently selected pose on the video canvas in to a pose embedding vector. Pose embeddings are 2D projections of pose skeleton data, where similar poses are embedded close to each other. This allows us to measure the similarity distance between two poses, akin to how word embeddings enable computing the distance between two words in NLP-based systems. We use Pr-VIPE [34] due to its robust performance in one-shot pose recognition. Most notably, the Pr-VIPE model was designed to be view-invariant, which means that pose effects created using PoseVEC would work with varying camera angles. The pose embedding vector is then stored internally in the recognizer node.

When the user switches to the online tracking mode, our system also computes a pose embedding vector for every new frame in the video window. When this data is available, all recognizer nodes currently in the node graph will perform a similarity check between the stored pose vectors and the current pose vector on the screen. We use L2 distance to estimate a similarity score. If the resulting score is smaller than a user-defined value, the pose recognizer node will continuously output true signals. This user-defined value serves as a threshold and is empirically set to three. The user can quickly fine-tune the threshold to his desired level of strictness by switching the video source or using a live webcam, while observing real-time changes.

4.2.2 Logic Node. Logic nodes can take pose recognizer nodes as inputs to construct conditional logics. Users can use logic nodes to create more interesting pose recognition behaviors. The input and output values of logic nodes are boolean values. There are two main categories of logic nodes: single-pose logic nodes and dual-pose logic nodes.

Single-pose logic nodes accept only one input. These nodes are used to create basic boolean operations. Specifically, trigger node outputs a single true value when the input value changes from false to true. It acts as a switch that is triggered by a change in the input value. Constant boolean node contains a toggle that outputs true (on) when the toggle is switched on, and false (off) when the toggle is switched off. Reverse logic node outputs the opposite value of its input.

Dual pose logic nodes typically take two or more pose recognizer nodes as inputs. Or node output true if any of the input is true. During node outputs true while the first input is true and until the second input is true. Sequence node outputs true only if both inputs had turned true sequentially. These nodes can be used to craft more complex behaviors. For example, with Or Node, users could chain multiple pose recognizer nodes together to create a more robust recognizer for a certain pose. During node is useful

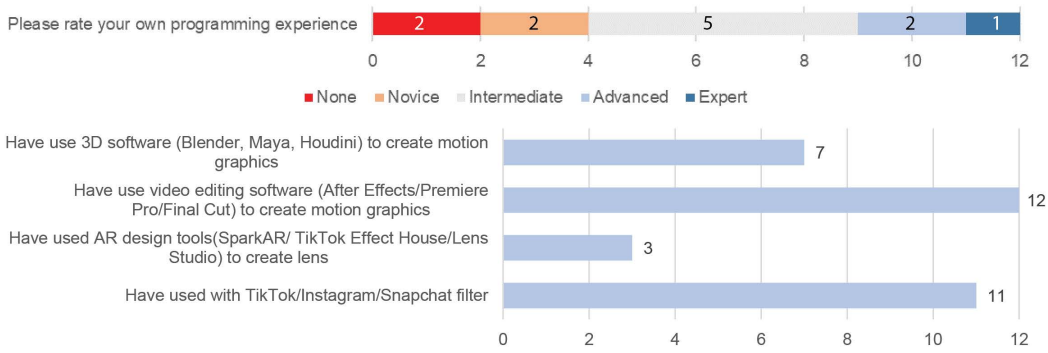

Figure 6: Distribution of participants' programming experience (top) and visual authoring experience (bottom).

for rendering animations that should only appear within a time range. And sequence node can be used for incremental tasks like counting.

4.2.3 Data Node. A data node is a complementary "source" to the pose recognizer node. It contains values useful for animating a motion graphics effect around the human body. These values can either be pose-related data or variables. The pose-related data could include joint locations, distances, angles or even pose embedding vectors. Data node actively monitors the skeleton pose on video canvas and computes output pose-related information. To create a pose-related data node, the user must first select the joints from the 2D estimated pose skeleton on the video canvas and then choose the type of data node the user wants to create. Additionally, variable nodes are used to store and manage numerical or vector data, and they can be directly added to the node canvas.

4.2.4 Transform Node. A transform node contains mathematical expressions and transforms the data in the data node into animation parameters. Some example operations supported by Transform nodes include standard arithmetic operations (plus, minus, multiply, divide), vector operations (L2Distance, scalar), conversions (normalization, number to text), and comparison.

4.2.5 Rendering Node & Graphical Assets. In PoseVEC, users can add by interacting with the asset panel(Figure 4, orange rectangle annotation). After adding, users can design and adjust the assets by dragging it on the video canvas. We provide options for working with primitive 2D shapes, texts, imported images and GIF animations. Users can customize their look (e.g., sizes, colors, and styles) or group assets together into a single object.

To render an asset, a user selects it on the video canvas and clicks on a render node. A render node will be added to the node canvas. A render node is a type of "sink" node, it only receives data from other nodes. Render nodes represent render functions that are executed on a group of graphical assets when receiving true signals.

PoseVEC provides different kinds of functions such as asset placement and movement, as well as adjusting asset opacity and color. In addition, PoseVEC provides special render nodes that contain a predefined asset group assigned with a specific render function. More importantly, it also exposes animation parameters as input for customization.

In particular, we define Joint Angle Annotation node and Joint Distance Annotation node, which are designed to compute joint-related information constantly and display that information on the human body directly. We also have a GIF animation render node for displaying and controlling GIF animation, and a Single Text Render node that accepts text input dynamically and displays that text.

A common rendering operation in pose effects is anchoring an asset to a pose joint in the video. PoseVEC simplifies this operation using video canvas interaction. When a new asset is added to the video canvas, by default, it is anchored to the video canvas. The user can anchor an asset to a joint on the pose skeleton by dragging and holding the asset for one second. A yellow line indicator will pop up to help users confirm the selected joint (Figure 5). Releasing the mouse click button will associate the joint identifier with the asset. After this assignment, any subsequent operations to set the position of the asset will record the position relative to the selected pose joint.

4.3 Implementation Details

We develop PoseVEC as a web application using Typescript. We use Fabric.js [30]. for canvas rendering and Animation.js [9]. for animation rendering. Our node graph model is based on Litegraph.js [5].

For ML capabilities, we use Tensorflow.js [2] and Mediapipe [20]. Specifically, we use Mediapipe to perform real-time 2D pose estimation on the incoming images from the video window. We support both uploaded video and live webcam video. We convert a pre-trained checkpoint of the Pr-VIPE model [34] into binary files that could then be loaded to the browser at run time. This model is then used to compute pose embedding vectors from the estimated pose skeleton data.

To improve playback and pose detection performance, we provide users with a script to pre-process the video file before uploading it to PoseVEC. The script computes pose skeleton and its corresponding pose embedding vector for each frame, and store all data into a JSON file. A user could then load both the video and the JSON file into our system.

5 USER EVALUATION

We conducted two studies to assess the usability and utility of PoseVEC. The first study was a first-use study with novice users to help us assess the system's usability and threshold. The second

study is a replicated example study to examine the utility and expressiveness of the system.

A more controlled experiment to compare PoseVEC with a baseline would be difficult because there is no clear baseline. PoseVEC focuses on pose-based expressions, which is not a common feature in video editing tools. Some commercial solutions like Lens Studio can support authoring body tracking based effects, but current workflow still requires a significant amount of programming knowledge. As our workflow design is geared toward designers, we wanted to first conduct this qualitative evaluation to assess the system's usability and examine the validity of our workflow for pose effect design. Although we did not test it with experts, our replicated example studies revealed some insights about what experts could achieve using our tool.

5.1 Participants

We recruited 12 participants to conduct our first-use study (5 males, 6 females, 1 prefer not to disclose; aging from 19 to 30). We selected users with some programming and visual effect creation experience (Figure 6). In general, most of our participants have some programming experience and have used AR lens filters. All of them have some experience in creating motion graphics using at least one type of commercial software.

5.2 Procedure

Each participant spent about two hours in the study. We began the session by presenting an overview of PoseVEC, followed by three tutorials of increasing difficulty for the participants to practice. For each tutorial, we showed the output effect of a node program and asked them to replicate the same effect. We then introduced the nodes that would be used for the tutorial and guided them through completing the example. The tutorial session lasted for about 50 minutes.

Open-Ended Exploration Session. In this session, we first showed our participants some pose-based effects that we created. We then asked them to describe a pose-based effect they would like to create using PoseVEC based on the examples we provided. During creation, participants could use the documentation of PoseVEC as a reference. They could ask questions about the system for clarification. When participants were satisfied with the results they created, they were asked to fill out the Likert scale questions (partially adapted from the SUS questionnaire [7]) as well as some open-ended questions. The session lasted about 60 minutes. We compensated the participants for their time.

5.3 Results and Discussion

Figure 8 shows an overview of the questions and the ratings given by participants for PoseVEC. All participants were able to explore PoseVEC and create unique pose-based effects. We show some examples in Figure 7. Please refer to the supplementary videos for more details on the participants' creations. In general, participants agreed that PoseVEC helped them create what they had in mind, (Mean = 4.67, SD = 0.49). P6 commented "*I am able to apply most of my imagination including using all the nodes introduced to me to create motion graphics*". P10 commented "*It has lots of flexibility in creation*".

Figure 7: Screenshot of pose effects created by participants using PoseVEC. Original videos ©Pexels

System Usability. In Q2 "I found the system very cumbersome to use" (Mean = 1.50, SD = 0.52), most participants agree that our system is not cumbersome to use because its UI is straightforward and similar to that of other software (like the blueprint in UnrealEngine), and its keyboard control is similar to others. They found the node programming and its operation useful because *"it visualizes variables and connections I have made for constructing motion graphics."* (P2). P4 commented that *"The operation like dragging assets and nodes to the scene is straightforward."*. P8 commented that *"The node programming makes the logic flow more modular. It breaks down into smaller steps so that you can modify to a higher degree."*

System Expressiveness. In Q4, "I feel like I could easily and quickly try out creative ideas by using this system." (Mean = 4.50, SD = 0.80) and Q5, "I feel like I could explore many alternative designs quickly by using this system." (Mean = 4.42, SD = 0.67), many participants agreed that they could try out creative ideas and explore many alternative design options by using PoseVEC. Several participants specifically mentioned that the use of pose recognizer helps them create design quickly. P10 commented that *"Normally I would use keyframes to create motion graphics, which I have to go back and forward to find the perfect keyframe in the video. But with this pose recognizer, I can drag the slider and add it to the scene, it is easier than my current workflow."*. P1 mentioned how easy it is to apply the same effect on different videos: *"I take a pose from one video and convert it to a pose recognizer, then I can use the same recognizer and nodes on other videos."* P3 said that the design of logic nodes is convenient. He was able to *"quickly construct complex behavior using existing logic nodes."*

Specially, all of the participants rated positively on how pose-related nodes like joint angle annotation and joint distance annotation nodes can speed up their creation process (Q7, Mean = 4.75,

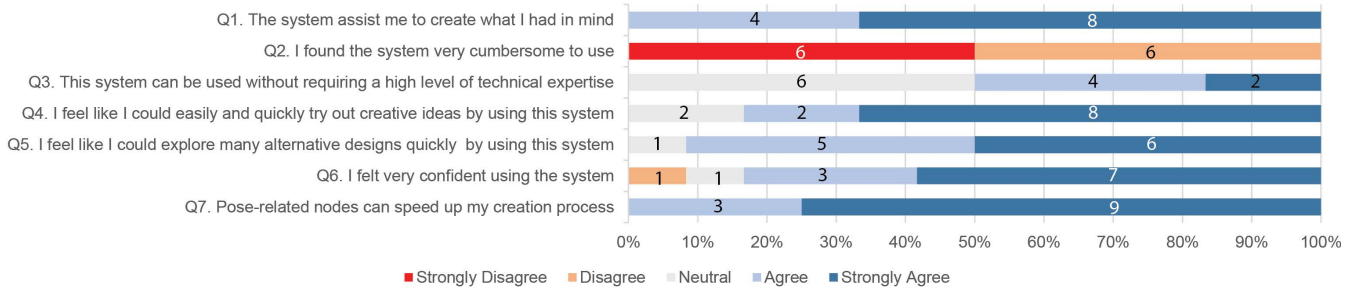

Figure 8: Percentage stacked bars showing the participants' ratings on PoseVEC after the open-ended exploration session.

SD = 0.45). P12 commented *"I can use joint distance annotation node to visualize the distance between joints, which helps me make decisions in designs*". P4 said *"It is nice because I don't need to do extra programming to implement it. I just add it to the scene and it is ready to go."* P2 commented *"I like it because once I define joint as input, this node handles the rest."*

Technical Expertise. When asked whether our system requires a high level of technical expertise (Q3, Mean = 3.67, SD = 0.78), half of our participants rated it positively while the other half had a neutral opinion. For participants who have a positive view on this, they said there is a learning curve for this system but the tutorials and documentation provided make it manageable. Those who had a neutral view pointed out that the logic nodes and data flow may not be common knowledge for someone with no programming experience. However, both sides agreed that people need to take some time to understand the concept and explore the system. The results of the first-use study showed that the majority of participants (10 out of 12) felt confident using our system (Q6, Mean= 4.33, SD=0.98). One participant described it as feeling natural to use, while another participant had a negative rating towards the system However, both of them mentioned that they needed more time to explore the system on their own to fully master it.

Workflow Comparison. In answering the question *"If you have to produce this result again, would you use your existing workflow (software), or would you consider using this system?*, 67% of the participants choose our system over their current workflow. Several participants who are relatively new to visual effects share a common belief that existing software is mainly designed for other types of functionality, which can be difficult for them to learn how to create the same effects. As a result, they prefer to use our system for expressive pose-based effects as they find the learning curve to be relatively mild.

Some participants who have more experience using visual effect editing software have different perspectives on this question due to some specific needs in their current workflow. One pointed out that *"Our system is more user friendly but it lacks some extra customization for graphical design. For showing pose-based related graphics this is the easiest one I have seen. but for creating other graphics, it lacks a bit of power" (P9).* P9 was referring to being able to create and customize more expressive animation such as spinning that our tool currently do not support well. P5 noted that since mainstream expressive AR lenses tend to focus on facial expressions, she prefers to use existing software for those purposes. However, for pose-based expressive lenses, she finds our tool to be more suitable since it is specifically designed for that purpose. P11 shared similar comments, *"If this tool can be integrated into the software I use, I definitely will use that to improve my workflow."*

Application of Pose Effects. In response to the question *"how would you use the output pose effect?"*, the majority of participants (10/12) mentioned that they would apply it to a workout training video. They believe that PoseVEC can help them improve their workout forms, prevent injuries as well as increase workout efficiency. They believe that the interactive visualization of poses can help people learn better. Some participants expressed interest in creating motion graphics for workout poses. Moreover, a few participants mentioned that PoseVEC can be used for animation creation and video editing. P12 said *"I would like to use it in video editing so that I can add effects on specific moments."* She said that she could use PoseVEC to combine her pose and some gif effects to create more expressive motion graphics.

In summary, the feedback from our novice users indicates that novice visual effect creators could learn the programming model of PoseVEC and use the system to produce the effect that they want. These findings also suggest the following key benefits of PoseVEC:

(1) Our approach of combining video interaction with pose recognition is straightforward and user-friendly.
(2) The specialized pose-based node design saves users from performing complex data transformations; instead, they could focus more on their creation process.
(3) Our visual programming workflow enables users to craft complex poses without requiring explicit programming.

5.4 Areas for Improvement

We also collected feedback from participants when they disagreed with the rating questions for future improvement.

UI Controls. Several participants commented on the need for UI improvements and better controls on asset properties. P10 suggested using icons to represent buttons: *"When I am doing design, I used to look for icons, not text buttons."* Participants also reported that it was sometimes inconvenient to scroll up and down between video canvas and node canvas to edit the nodes and see the effect: *"Not being able to increase the size of node canvas makes it difficult to read and do operation node canvas."* (P9). They suggested that adding a customization for the node canvas window can solve this issue. Moreover, they also suggested to add a variety of styles and

Figure 9: Keyframes of the three replicated examples: YouMove [6] Yoga Tutorial (top), PoseTween [19] Basketball (middle), Kinetic Typography (bottom). Original videos ©Pexels

properties to current assets such as gradient color (P12, P3) and color filters (P8).

Animation and Workflow. Several participants recommended PoseVEC should add more render nodes to enable more precise control over animation movement. They requested functionalities to trace animation paths using brushes (P1) or joint movements (P12) as paths that can be edited later on. This would give them greater control and flexibility in designing movement animations.

6 REPLICATED EXAMPLE STUDY

To show the utility and expressiveness of the authoring workflow in PoseVEC, we conducted a replicated example study and recreated three complex pose effects from examples. We followed the Type 1 Demonstration study approach that Ledo et al. had characterized [17]. The first example is an interactive Yoga tutorial inspired by the YouMove system [6]. The second example is an interactive basketball dribbling effect inspired by the PoseTween system [19]. The third example called "Just Do It" is a kinetic typographic effect inspired by a collection of effect we found online [27]. Figure 9 shows some keyframes of the examples that we produced. The full demonstrations of these replications can be found in the supplementary video.

6.1 YouMove's Yoga Tutorial

The goal of this example is to showcase how PoseVEC can be used to create interactive exercise tutorial content. A unique feature of the YouMove system is pose guidance. Using an augmented mirror, a user could receive real-time feedback about her own pose when striking that pose for a period of time in front of the system. The system could also provide feedback to the user's performance with some on-screen texts.

To reconstruct this key feature, we first use the elapse time data node and the comparison transform node to create a time delay trigger. The trigger will fire True signals after the user strikes a pose

for a period of time. Connecting to this delay trigger, we further design the pose guidance feedback text rendering mechanism. To do this, we need to convert pose similarity scores to performance rating text. We obtain current pose information on the video and then compute similarity scores with a target pose using a combination of pose-related data nodes and a vector math node. We then normalize the scores to values between zero and one using a transform node, and convert them into performance text using a "number to string" transform node. This performance text serves as input for the render node. With this program, the text can be rendered while the user was holding the pose, providing real-time feedback on the similarity between the user's current pose and a target pose of a Yoga instructor.

6.2 PoseTween's Ball Dribbling

This example aims to demonstrate the ability of PoseVEC in creating interactive animations driven by animation parameters. The key challenge of PoseTween is to recreate the basketball dribbling and shooting effects. We need to create a hand-bouncing and hand-shooting behaviors for the virtual ball.

For the first effect, the ball should exhibit a vertical movement that corresponds to the user's hand. The movement of the ball must also synchronize with the up-down movement of the person's hand movement. To allow users to control the ball movement with their hands, we used a joint angle data node to capture the angle of the user's elbow joint. This data is then normalized into a value between zero and one using a transform node so that it can be used to control the animation timeline of the ball. We uploaded a basketball dribbling GIF animation as a render node. Then we anchored the GIF animation to a desired location near to the hand joint. We then connected the aforementioned transform node to the GIF animation rendering node, so the elbow angle in the person pose can be used to drive the timeline of the GIF animation. As a result, when the user raises his hand (i.e., the elbow angle is low), the animation seeks to the beginning (near the hand); and when the user lowers his hand(i.e., with a high elbow angle), the animation seeks to the end (e.g., near the floor). Finally, we used a change opacity node to hide this render node when the user performs the shooting pose by connecting it to the shooting pose recognizer.

To ensure that the first effect only occurs before the shooting effect, we added a new pose recognizer for the ball-shooting pose, and then used a during node that connects the ball-dribbling pose recognizer and the ball-shooting pose recognizer. For the shooting effect, we also uploaded a basketball shooting GIF animation as a render node and anchored it to the user's hand. To ensure that the user only saw this animation when the he was performing a shooting ball pose, we used a change opacity node to hide this animation when the ball-dribbling pose occurred. We then connected the shooting pose to a ball-shooting animation.

6.3 "Just Do It" Kinetic Typography

In the original collection, the artists at Shotopop studio created many different kinetic typography effects where the text "Just Do It" are stylized and animated dynamically to various athletic poses. We attempted to create an example in the style of this collection. Briefly, for this effect: 1) the texts would initially show up at the user's feet and gradually move to the user's shoulders; 2) as the user performs a jumping jack pose motion, we would dynamically stylize the text by changing its color and line distance parameters. There are two main tasks involved in creating this example: animating the color, opacity, and positions of the text and implementing logical conditions to control the changes in response to the jumping jack motion.

For the first task, we combined the recognizer node with the during node to ensure that the computation occurred only when users were performing a jumping jack pose. Then we obtained the joint location of a wrist from the data node. By computing the height of the wrist location between two jumping poses, we converted it into an animation parameter that controlled animation render nodes.

For the second task, we use the moveTo render node to render keyframed transformations of the text. This node allows us to record a beginning state and ending state for the input text, and it will render the interpolated changes in between these two states. We made the size and the line distance slightly bigger for the end state to create the expanding effect for the text. We then anchored the text to the user's feet as the starting location of the movement and then to the user's shoulder as the ending location. Additionally, we also added the change opacity node for the appearing animation, and the change color node to alter the text's color.

7 DISCUSSION

7.1 Applications Beyond Social Media Videos

Many examples in our current paper draw inspiration from social media themes like dances and sports. Although not extensively tested, PoseVEC can also be used for other applications such as how-to videos, kinesiology assessments, or even rehabilitation exercises.

The essence of how-to videos is providing step-by-step instructions for users to follow along. With PoseVEC, designers can create an interactive how-to video that supports users learning at their own pace and provides real-time feedback. For example, in a how-to-squat video, each step of squatting can be represented by a pose recognizer node, and a Sequence node can be used to represent the transition between steps. More importantly, key joints such as knee joints can be highlighted to emphasize a correct pose movement and avoid injuries. Also, users' movement can be tracked and visualized on screen in real-time, and their performance can be evaluated. All of these can be done by using a combination of Joint Data node, transform node and render node.

Kinesiology assessments focus on evaluating human movement, strength, and flexibility. Adding visual effects on top of these assessments can help individuals identify their weaknesses and understand their movement patterns better. For example, a sit-and-reach test can be constructed similar to the pose program from YouMove's yoga tutorials. Instead of guiding users to follow a yoga pose, we ask them to follow a sit-and-reach pose. We would use a Joint Distance node to measure the distance between the hands and the feet, and compute performance scores with a target distance with transform nodes. Then, we can display the real-time performance result on the video screen while the user is holding that pose. Future work in this direction could consider adding more nodes tailored for kinesiology assessment such as measuring body flexibility.

Overall, the combination of PbD with direct manipulation approach in PoseVEC facilitates a streamlined authoring workflow. This allows users to focus on their creation of pose-aware visual effects and apply them to a variety of motion-driven applications.

7.2 Limitations and Future Work

More Complex Visual Effects. Our work suggests several interesting directions for future research. The field of pose-aware visual effects has a rich design space. We would like to incorporate depth estimation [16] into our current workflow to create more pose-aware graphics according to changes in the pose size. For example, as a user walks away from the camera, the graphic could resize accordingly. Moreover, it is worth exploring a combination of multiple techniques such as scene segmentation and continuous motion tracking to create more complex effects such as rotoscoping. PoseVEC is currently limited to prototyping single-person pose effects, it would be valuable to extend the workflow to explore group interactions such as collaborations and competitions. To enable this, PoseVEC has to incorporate a multi-person pose tracking technique [11] to replace the current Mediapipe backend. In addition, creating pose effects for groups would require the use of new node designs. For instance, we could create a group exercise counter which counts as the entire group completes a set of exercises. Alternatively, we could create an animated text effect that responds to the distance between two dancers on screen.

Multimodal ML-based Recognizers. Although PoseVEC focuses specifically on pose-effect authoring, the framework in PoseVEC could potentially generalize to authoring effects based on other types of discrete ML-based recognizer like sound, effect, facial gesture, or hand gesture. A direction for future work is to explore extending PoseVEC into a general authoring tool for multimodal ML-based recognizers.

Mistake Handling in PoseVEC. PoseVEC currently leverages an off-the-shell pose embedding model called Pr-VIPE [34] to handle pose matching. This model has been shown to be effective against view angle changes and self-occlusion. However, false positives might still occur. PoseVEC allows users to tune the behavior of the recognizer using threshold or by combining multiple recognizer nodes together. However, since we are focusing on the authoring aspect of PoseVEC, we have not evaluated it in a production use case where false positives are much more difficult to control. Future work should consider how to fine-tune or even retrain Pr-VIPE on more domain-specific pose dataset (e.g., dance) to enable export and production workflow in PoseVEC. Another possibility for future extension is to integrate pose editing techniques such as those in PoseTween[19] to correct for bad pose detection in user videos.

Portability Assessment. One benefit of PoseVEC node program is that a resulting node program could be re-applied to another video. Although we did not formally assessed this portability benefit, we demonstrated it in two examples in Section 6. Specifically, We applied the "Just Do It" (Sec. 6.3) effect to other videos with different subjects performing the same movement and applied the "Ball Dribbling" (Sec. 6.2) effect to another video with the same subject performing a slightly different movement. We found that in the former, effect transfer works well because the animation parameters were defined solely based on pose joint data (i.e., Just Do It). However, in the latter example, when the parameters depend on non-joint data such as the distance between a person's hand and the ground floor, more adjustments are needed to transfer the effect to a new video. These empirical findings suggest that with user adjustments, PoseVEC effects can be applied to videos with similar movements. To improve effect portability, future work could focus on automatically inferring and updating animation parameters based on multiple examples, similar to the Gamut system [23].

Limitations of PbD approaches and UIs. A common drawback of PbD approaches is that they do not support low-level customization well. Our system does not suffer from this drawback since users can use nodes to customize the behavior of an effect. However, this level of customization depends on the set of available nodes. In our user evaluation, we found that more experienced users usually requested richer node functionalities like animation design and facial expression that PoseVEC is not currently supporting. Feedback from users in our user study suggests that there is still room for improvement in the user interface of our tool. For instance, when making connections between nodes, it would be helpful to show hints or suggestions about which nodes could be connected. This would prevent non-technical users from making mistakes. While PoseVEC has an inspector panel that lists all graphical assets and their associated render nodes, it would be beneficial to allow users to preview the animation details of each render node, making it easier to modify each render node. It would be helpful to provide a library of node programs and templates for designers to browse for ideas and improvise.

Evaluation Limitations. Our approach aims to make pose effect authoring easy and accessible to a broader audience through PbD and visual programming. Our current user evaluation, however, only examines the immediate usability and intuitiveness of our approach. Understanding the true potential utility of the tool or how it fares with other tools require longitudinal evaluation, and remain as future work.

8 CONCLUSION

We present PoseVEC, a lightweight web-based authoring tool to create expressive and adaptive pose effects. To flatten the learning curve for users, it provides a visual workflow using node programming. It combines programming by demonstration and visual programming to allow users to create pose recognizer directly from the input video stream, obtain low-level pose information from node programs, and easily convert it into animation parameters. More importantly, it also relieves the burden of non-technical designers for testing. The visualization of node programming allows users to quickly test and refine the pose effect program, making it easier for them to iterate and improve their designs. We evaluated PoseVEC's usability and utility through two stuides: first-use study with novice users and a replicate example study. All users from the first-use study could create expressive pose effects efficiently.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their constructive comments. This research is supported by an NSF Graduate Research Fellowship and an NSF CAREER Award (award number: 1942531). We are also thankful for Adobe Research's support to the GMU's

DCXR Lab. We would also like to thank Ana Cardenas Gasca for providing initial feedback on our early-stage prototypes.

REFERENCES

[1]]adobepremierepro [n. d.]. Professional video editing software | adobe premiere pro. https://www.adobe.com/products/premiere.html

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[3] Adobe. 2023. Create 3D Models on Desktop and in VR. https://www.adobe.com/products/substance3d-modeler.html. [Accessed 04-Apr-2023].

[4] Adobe. 2023. Motion graphics software | Adobe After Effects — adobe.com. https://www.adobe.com/products/aftereffects.html. [Accessed 05-Apr-2023].

[5] Javi Agenjo. 2022. litegraph.js. https://github.com/jagenjo/litegraph.js?files=1. [Accessed 05-Apr-2023].

[6] Fraser Anderson, Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2013. YouMove: Enhancing Movement Training with an Augmented Reality Mirror. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology* (St. Andrews, Scotland, United Kingdom) *(UIST '13)*. Association for Computing Machinery, New York, NY, USA, 311–320. https://doi.org/10.1145/2501988.2502045

[7] John Brooke. 1995. SUS: A 'Quick and Dirty' Usability Scale, Usability Evaluation in Industry.

[8] Jiawen Chen, Shahram Izadi, and Andrew Fitzgibbon. 2012. KinÊtre: Animating the World with the Human Body. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (Cambridge, Massachusetts, USA) *(UIST '12)*. Association for Computing Machinery, New York, NY, USA, 435–444. https://doi.org/10.1145/2380116.2380171

[9] Julian Garnier. 2023. anime.js. https://animejs.com/. [Accessed 05-Apr-2023].

[10] Epic Games Inc. 2023. Unreal Engine. https://www.unrealengine.com/. [Accessed 04-Apr-2023].

[11] Umar Iqbal, Anton Milan, and Juergen Gall. 2017. PoseTrack: Joint Multi-person Pose Estimation and Tracking. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4654–4663. https://doi.org/10.1109/CVPR.2017.495

[12] Jennifer Jacobs, Joel Brandt, Radomír Mech, and Mitchel Resnick. 2018. Extending Manual Drawing Practices with Artist-Centric Programming Tools. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3174164

[13] Amir Jahanlou and Parmit K Chilana. 2022. Katika: An End-to-End System for Authoring Amateur Explainer Motion Graphics Videos. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 502, 14 pages. https://doi.org/10.1145/3491102.3517741

[14] Brian Jordan, Nisha Devasia, Jenna Hong, Randi Williams, and Cynthia Breazeal. 2021. PoseBlocks: A Toolkit for Creating (and Dancing) with AI. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 17, 15551–15559. https://doi.org/10.1609/aaai.v35i17.17831

[15] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. 2012. Proton: Multitouch Gestures as Regular Expressions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2885–2894. https://doi.org/10.1145/2207676.2208694

[16] J. Kopf, X. Rong, and J. Huang. 2021. Robust Consistent Video Depth Estimation. , 1611-1621 pages. https://doi.org/10.1109/CVPR46437.2021.00166

[17] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3173574.3173610

[18] Germán Leiva, Jens Emil Grønbæk, Clemens Nylandsted Klokmose, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. 2021. Rapido: Prototyping Interactive AR Experiences through Programming by Demonstration. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 626–637. https://doi.org/10.1145/3472749.3474774

[19] Jingyuan Liu, Hongbo Fu, and Chiew-Lan Tai. 2020. PoseTween: Pose-Driven Tween Animation. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 791–804. https://doi.org/10.1145/3379337.3415822

[20] Google LLC. 2022. MediaPipe — mediapipe.dev. https://mediapipe.dev/. [Accessed 05-Apr-2023].

[21] Hao Lü and Yang Li. 2012. Gesture Coder: A Tool for Programming Multi-Touch Gestures by Demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2875–2884. https://doi.org/10.1145/2207676.2208693

[22] Hao Lü and Yang Li. 2013. Gesture Studio: Authoring Multi-Touch Interactions through Demonstration and Declaration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 257–266. https://doi.org/10.1145/2470654.2470690

[23] Richard G. McDaniel and Brad A. Myers. 1999. Getting More out of Programming-by-Demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) *(CHI '99)*. Association for Computing Machinery, New York, NY, USA, 442–449. https://doi.org/10.1145/302979.303127

[24] George B. Mo, John J Dudley, and Per Ola Kristensson. 2021. Gesture Knitter: A Hand Gesture Design Tool for Head-Mounted Mixed Reality Applications. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 291, 13 pages. https://doi.org/10.1145/3411764.3445766

[25] Kyzyl Monteiro, Ritik Vatsal, Neil Chulpongsatorn, Aman Parnami, and Ryo Suzuki. 2023. Teachable Reality: Prototyping Tangible Augmented Reality with Everyday Objects by Leveraging Interactive Machine Teaching. , Article 459 (2023), 15 pages. https://doi.org/10.1145/3544548.3581449

[26] Brad A. Myers, Richard McDaniel, and David Wolber. 2000. Programming by Example: Intelligence in Demonstrational Interfaces. *Commun. ACM* 43, 3 (mar 2000), 82–89. https://doi.org/10.1145/330534.330545

[27] NIKE OLYMPICS. 2022. NIKE OLYMPICS. https://www.behance.net/gallery/73353531/NIKE-OLYMPICS. [Accessed 05-Apr-2023].

[28] Randy Pausch, Tommy Burnette, AC Capeheart, Matthew Conway, Dennis Cosgrove, Rob DeLine, Jim Durbin, Rich Gossweiler, Shuichi Koga, and Jeff White. 1995. Alice: Rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications* 15, 3 (1995), 8–11. https://doi.org/10.1109/38.376600

[29] petricoregames. 2023. Pose Dancer. https://lenslist.co/pose-dancer/. [Accessed 05-Apr-2023].

[30] Printio.ru Lab Project. 2023. Fabric.js Javascript Canvas Library. http://fabricjs.com/. [Accessed 05-Apr-2023].

[31] Nazmus Saquib, Rubaiat Habib Kazi, Li-Yi Wei, and Wilmot Li. 2019. Interactive Body-Driven Graphics for Augmented Video Performance. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300852

[32] Yeongho Seol, Carol O'Sullivan, and Jehee Lee. 2013. Creature Features: Online Motion Puppetry for Non-Human Characters. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Anaheim, California) *(SCA '13)*. Association for Computing Machinery, New York, NY, USA, 213–221. https://doi.org/10.1145/2485895.2485903

[33] ShaderPlayStudios. 2014. ShaderPlay.com. https://www.shaderplay.com/. [Accessed 04-Apr-2023].

[34] Jennifer J. Sun, Jiaping Zhao, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, and Ting Liu. 2020. View-Invariant Probabilistic Embedding for Human Pose. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V* (Glasgow, United Kingdom). Springer-Verlag, Berlin, Heidelberg, 53–70. https://doi.org/10.1007/978-3-030-58558-7_4

[35] William Robert Sutherland. 1966. *The on-line graphical specification of computer procedures.* Ph. D. Dissertation. Massachusetts Institute of Technology.

[36] Ryo Suzuki, Rubaiat Habib Kazi, Li-Yi Wei, Stephen DiVerdi, Wilmot Li, and Daniel Leithinger. 2020. RealitySketch: Embedding Responsive Graphics and Visualizations in AR with Dynamic Sketching. In *Adjunct Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20 Adjunct)*. Association for Computing Machinery, New York, NY, USA, 135–138. https://doi.org/10.1145/3379350.3416155

[37] Tianyi Wang, Xun Qian, Fengming He, Xiyun Hu, Yuanzhi Cao, and Karthik Ramani. 2021. GesturAR: An Authoring System for Creating Freehand Interactive Augmented Reality Applications. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 552–567. https://doi.org/10.1145/3472749.3474769

[38] Lei Zhang and Steve Oney. 2020. FlowMatic: An Immersive Authoring Tool for Creating Interactive Scenes in Virtual Reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 342–353. https://doi.org/10.1145/3379337.3415824

A APPENDIX

Table 1 shows all nodes in PoseVEC. We also identify "video-linked" node, source node, and sink node in the table.

Node Type	Subcategory	Node Name	Description	source	video-linked	sink
Recognizer		Pose Recognizer	It takes the current pose from the video canvas as key pose; When online tracking is enabled, it constantly compares the key pose with poses from the video canvas, and shows a similarity score. It outputs a true signal when the similarity the score is below a user-defined value.	X	X	
Logic node	Single Pose Logic	Constant boolean	It contains a toggle that outputs true (on) when the toggle is switched on, and false (off) when the toggle is switched off.			
		Trigger	It outputs a single true value when the input value changes from false to true. It acts as a switch that is triggered by a change in the input value.			
		Reverse logic	It outputs the opposite value of its input.			
		Turn on once	Special version of trigger node, only activate once during online tracking mode			
	Dual Pose Logic	Or	It outputs true if any of the input is true. Users could chain multiple pose recognizer nodes together to create a more robust recognizer for a certain pose.			
		Sequence	It outputs true only if both inputs had turned true sequentially.			
		During	It is useful for rendering animations that should only appear within a time range. It outputs true while the first input is true before the second input turns true.			
Data node	Joint Information	Joint indices	It outputs selected joint indices.	X	X	
		Joint location	It outputs selected joint location.	X	X	
		Joint angle	It outputs selected joint angle.	X	X	
		Joint distance	It outputs selected joint distance.	X	X	
	Pose Information	Current pose embedding	It outputs embedding vectors of poses from video canvas.	X		
		Target pose embedding	It outputs embedding vectors of pose from user-defined pose recognizer.	X		
	Variable	Constant number	It outputs a constant number.	X		
		Numeric variable	It outputs a number variable. Users must specify if they want to update this variable during the online tracking stage. If they do, users must specify the transform node ID, which is the execution point after which the variable will be updated.	X		
		Vector variable	Same as above. Instead of outputting number variable, it outputs vector variable.	X		

Table 1

Node Type	Subcategory	Node Name	Description	source	video-inked	sink
Transform node	Basic Math	Basic math	+,-,*,/ operations			
		Vector math	+,-, L2 distance operations			
		Vector scalar	*,/ operations			
		Convert to [0,1]	normalization			
	Comparison	Compare node	Compare two numeric values using a user-defined comparator such as greater than or equal to (>=) and less than or equal to (<=).			
		Location comparison	Compare two vector variable A and B using a user-defined comparator. For example, outputs true if A is on the left of B.			
	Other Function	[0,1]To text	Convert input value to text based on its range. For example, user can define if x < 0.3, output 'bad'; if x >= 0.3, output 'excellent'.			
		Elapse time	Record and output theamount of time that has passed when the input is true.			
Render node	Basic	Place at animation	It renders the appearance animation of a group of graphical objects at a specific location.		X	X
		Chang opacity animation	It controls the opacity of a group of graphical objects.		X	X
		Move To animation	It moves a group of graphical objects from one place to another.		X	X
		Change color animation	It changes color of a group of graphical objects.		X	X
	Special Render	Joint Angle Annotation	It displays the angle of a joint as text and highlights it using two lines and an arc.			X
		Joint Distance Annotation	It displays the distance between two joint as text and highlights the joints by a line.			X
		Single Text Render	It accept a number, vector, or text as input and display the content using a text object.			X
		Gif Render Node	It renders a GIF animation. It contains properties such as animation timeline, scale, and placement, which can be modified by other nodes.			X