Generating Activity Snippets by Learning Human-Scene Interactions

CHANGYANG LI, George Mason University, USA
LAP-FAI YU, George Mason University, USA

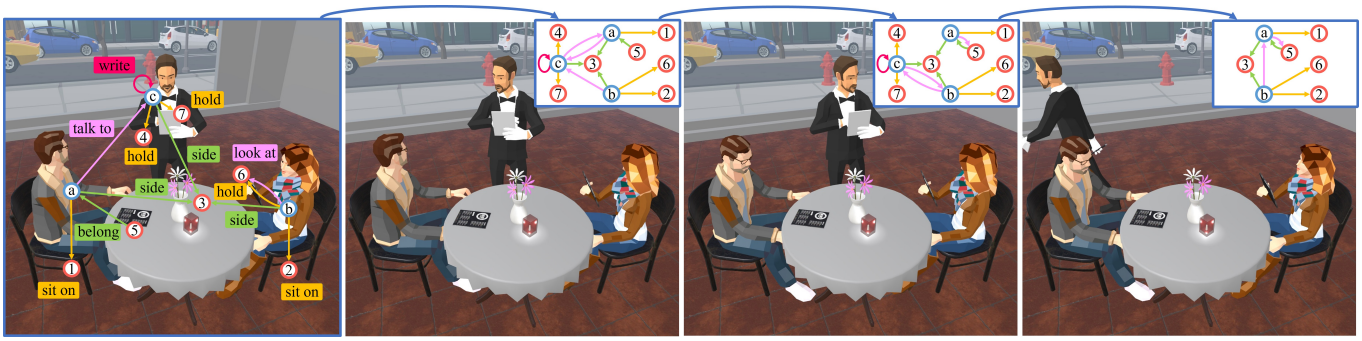

Fig. 1. Our approach generates an activity snippet referring to a sequence of high-level keyframe descriptions of multi-character, multi-object interactions represented as graphs, based on which our approach automatically instantiates virtual characters and objects in a 3D environment to illustrate an activity. This example shows a generated activity snippet of a waiter taking orders from two customers in a restaurant.

We present an approach to generate virtual activity snippets, which comprise sequenced keyframes of multi-character, multi-object interaction scenarios in 3D environments, by learning from recordings of human-scene interactions. The generation consists of two stages. First, we use a sequential deep graph generative model with a temporal module to iteratively generate keyframe descriptions, which represent abstract interactions using graphs, while preserving spatial-temporal relations through the activities. Second, we devise an optimization framework to instantiate the activity snippets in virtual 3D environments guided by the generated keyframe descriptions. Our approach optimizes the poses of character and object instances encoded by the graph nodes to satisfy the relations and constraints encoded by the graph edges. The instantiation process includes a coarse 2D optimization followed by a fine 3D optimization to effectively explore the complex solution space for placing and posing the instances. Through experiments and a perceptual study, we applied our approach to generate plausible activity snippets under different settings.

CCS Concepts: • **Computing methodologies** → **Graphics systems and interfaces**.

Additional Key Words and Phrases: graph generation, behavior synthesis, character animation, mixed reality

ACM Reference Format:
Changyang Li and Lap-Fai Yu. 2023. Generating Activity Snippets by Learning Human-Scene Interactions. *ACM Trans. Graph.* 42, 4, Article 1 (August 2023), 15 pages. https://doi.org/10.1145/3592096

Authors' addresses: Changyang Li, George Mason University, USA, cli25@gmu.edu; Lap-Fai Yu, George Mason University, USA, craigyu@gmu.edu.

0730-0301/2023/8-ART1
https://doi.org/10.1145/3592096

1 INTRODUCTION

Generating virtual environments and compatible human behaviors is useful for diverse tasks in computer graphics, computer vision and robotics. With the availability of large-scale datasets, deep generative methods have become popular for learning from real-world recordings and generating novel samples.

The recent trends of 3D animation, simulation, and VR/AR applications such as the metaverse show a substantial demand for generating plausible virtual activities pertaining to virtual characters' and objects' placements and the interactions among them [Hassan et al. 2021b; Zhang et al. 2020c,a]. Existing methods may support posing single or multiple virtual humans in a given virtual environment, but are not guided by high-level contexts such as what activities the virtual humans perform. Some other works synthesize motions [Hassan et al. 2021a; Starke et al. 2019; Wang et al. 2021a,b], which are usually correlated with specific activity labels. However, most prior works consider a single virtual human's interactions with a static scene only.

In this paper, we study a relevant problem of generating a sequence of complex multi-character, multi-object interactions, considering the tangled spatial-temporal and interactive relations. Our goal is to generate virtual **activity snippet**s, which depict sequenced multi-character, multi-object interaction scenarios in virtual 3D environments, by learning from real human-scene interaction recordings. Figure 1 shows an example of a "take orders" activity snippet in a restaurant, which contains four keyframes. In this example, two customers hold their menus and sit by a table, and a waiter jots down their orders and leaves the scene. The graphs, dubbed as **keyframe description**s, encode abstract interactive contexts in the scene. Nodes and edges in a keyframe description are associated with certain categories to help represent the activity. Such abstract interactions are eventually instantiated into 3D environments.

In comparison to prior works that study how to use data-driven models to generate detailed human poses or motions (i.e. kinematic

parameters of body parts and joints), our work focuses on the high-level planning of symbolic interactions (e.g., a character sits on a chair and talks to someone) in a deep generative manner, and then assigns appropriate 3D poses to characters and objects accordingly. Specifically, we use a sequential graph generative model to capture the transitions of interactions between adjacent keyframe descriptions, guided by learned temporal relations using a temporal module. Our approach can then generate a keyframe description sequence, which incorporates abstract interactions in an activity snippet, by repeatedly triggering the keyframe description generator. The temporal module also decides when the generation should appropriately terminate for a scenario. In order to instantiate activity snippets from high-level abstractions to concrete 3D placements, we devise an optimization framework, which consists of a coarse 2D optimization with a progression schema and a fine 3D optimization, to effectively explore the complex solution space comprising a varying number of constraints derived from edges in keyframe descriptions.

Our approach can be applied for compelling use cases. For example, it can be applied to generate novel non-player character (NPC) behaviors in dynamic virtual environments using learned human-scene interactions instead of hand-crafted rules or behavior trees. It may also be used to predict appropriate NPC behaviors in response to players' behaviors. Our approach can help create faithful social scenarios in the metaverse; and in turn, complex multi-character, multi-object interactions recorded from real users in 3D metaverse environments will facilitate the training of our approach for generating more plausible activities.

In summary, our contributions include:

- Introducing the novel concept of *activity snippets* to represent sequential multi-character, multi-object interactions in 3D virtual environments.
- Proposing a sequential deep graph generative model to generate keyframe descriptions to depict an activity, guided by spatial-temporal and interactive relations learned from human-scene interactions.
- Devising a two-stage optimization framework to realize keyframe descriptions as 3D instance placements while considering a variety of constraints among characters and objects in the scene.

2 RELATED WORK

2.1 Realizing Human Behaviors in 3D Environments

Generating scene-aware virtual human behaviors is a long-standing problem in graphics and vision. A representative problem is generating virtual human poses adaptively in input scenes. For example, Kim et al. [2014] predict poses by analyzing the correlations between contact points and geometric features of shapes. Savva et al. model correlations between geometries and functionalities of 3D environments [Savva et al. 2014], and correlations between human poses and object arrangements [Savva et al. 2016], using observations of from human-scene interactions. Recent advances in deep learning techniques and the increasing availability of 3D data stimulate research on posing virtual humans considering geometries, semantics, affordances, and contacts [Hassan et al. 2021b; Zhang et al. 2020c,a] using deep generative models.

In addition to generating static poses, scene-aware human motion generation should cope with temporal relations and interactions between humans and scenes, and should be guided by tasks in specific scenarios. For high-level planning of motion, Lee et al. [2006] use building blocks to allow navigation of animated characters. Bai et al. [2012] generate manipulation sequences by traversing manipulation graphs. Agrawal et al. [2016] use task-specific foot-step plans as motion templates. Pirk et al. [2017] present interaction landscapes as functional descriptors for proximal interactions. Lee et al. [2022] present a framework to create physically compliant interactions with surroundings for humanoids. Recent learning-based methods address the problem of generating motions as sequences of parameterized human poses [Cao et al. 2020], and may further be conditioned on surrounding environments [Hassan et al. 2021a; Starke et al. 2019; Wang et al. 2021a,b], action labels [Guo et al. 2020; Petrovich et al. 2021], or textual descriptions [Athanasiou et al. 2022; Ghosh et al. 2021; Wang et al. 2022]. Script-based animation may find applications in gaming and storytelling. For example, Text2SceneVR [Abrami et al. 2020] is a system that allows users to create VR scenes using hypertexts. Zhang et al. [2021] convert scripts as semantic scene graphs for creating animations. ASAP [Kim et al. 2021] parses scripts into descriptions of actions, characters and dialogues for animation synthesis. Nawmal [Nawmal 2019] is a commercial tool for text-based animation/scene creation. Some recent works also investigate how to accommodate virtual humans in real environments for augmented reality applications [Li et al. 2022; Tahara et al. 2020].

Our work focuses on high-level symbolic planning of dynamic multi-character multi-object activities in temporal keyframes, and optimizes the poses of characters and objects guided by abstract activity graph descriptions generated by a deep graph generative model. Related to our problem, PiGraphs [Savva et al. 2016] focuses on detailed stickman poses considering spatial relationships but not temporal interactions. Wang et al. [2019b] investigate synthesizing animation sequences of interactions between a single human-hand and multiple foreground objects, assuming the number of objects is unchanged.

While a large portion of works realizes human behaviors in captured real-world environments, a related topic is synthesizing virtual layouts for human activities or character animations. Early solutions include optimizing cost functions derived from interior design principles [Merrell et al. 2011; Yu et al. 2011], and learning statistical models or graph models for modeling layout patterns [Fisher et al. 2012; Fu et al. 2017]. More recently, learning-based approaches are devised based on large-scale 3D scene datasets for layout synthesis [Hu et al. 2020; Sun et al. 2022; Wang et al. 2019a, 2018; Zhang et al. 2020b]. On the other hand, human behaviors can also guide 3D scene synthesis [Fisher et al. 2015; Ma et al. 2016; Ye et al. 2022]. In our work, we synthesize poses of both characters and objects in all keyframes jointly using an optimization framework.

2.2 Complex Human Activity Datasets

A large number of human activity datasets are available to support various activity-related tasks. For example, datasets like ActivityNet [Caba Heilbron et al. 2015], THUMOS [Idrees et al. 2017],

FineGym [Shao et al. 2020] and AVA-Kinetics [Li et al. 2020] facilitate action recognition research, which refers to identifying events performed by humans. Action Genome [Ji et al. 2020] represents human-object relationships as spatial-temporal scene graphs. LEMMA [Jia et al. 2020] contains recordings of complex multi-agent, multi-task human activities with atomic action labels. Recently, Luo et al. present MOMA [Luo et al. 2021] for activity parsing and MOMA-LRG [Luo et al. 2022] as a follow-up work to incorporate natural language descriptions. In our work, we utilize the MOMA [Luo et al. 2021] dataset, which contains the annotations of multi-object multi-actor activities encoded by hypergraphs, to learn complex human-scene interactions for generating novel activity snippets.

2.3 Deep Generative Models of Graphs

According to a recent survey [Guo and Zhao 2022], deep graph generation methods, including conditioned or unconditioned generations, can be broadly categorized as: (1) one-shot generation methods, which generate graphs by sampling from the probabilistic distribution of graph latent space in one step, such as GraphVAE [Simonovsky and Komodakis 2018] and JT-VAE [Jin et al. 2018]; and (2) sequential generation methods, which generate a graph by making sequential decisions of adding nodes and edges, conditioning on the sub-graphs already generated, such as DGMG [Li et al. 2018] and GraphRNN [You et al. 2018]. Since our approach repeatedly creates a new graph for the next frame conditioned on the graph of the previous frame, graph modifications are frequently required and thus sequential generation fits our case well. We employ a sequential generative model based on DGMG [Li et al. 2018] and incorporate node deletion and edge deletion functions to tackle the problem of activity snippet description generation discussed in Section 4.1.

3 OVERVIEW

Our approach synthesizes a virtual activity snippet by first generating a sequence of keyframe descriptions, which depict high-level interactions in a scene with multiple characters and objects, and then instantiating the 3D placements through optimizations. Figure 2 shows an overview of our approach.

Generation of Keyframe Descriptions. Given an activity label, our keyframe description generator repeatedly generates new keyframe descriptions, represented by graphs and conditioned on their predecessors. In the end, a sequence of keyframe descriptions that incorporate dynamic interactions is formed to satisfy the desired activity (Section 5). As shown in the upper part of Figure 2, a keyframe description at frame k, together with an up-to-date temporal signal produced by a recurrent model (LSTM in our implementation), are passed to the generator for generating a new description for frame $k+1$. The keyframe description generator is implemented as a sequential generative model, which utilizes message-passing graph convolution to learn abstract relationships encoded in graphs.

Activity Snippet Instantiation. The generated keyframe descriptions in the activity snippet are used to guide an instantiation model to synthesize plausible 3D instance placements (Section 6). Specifically, nodes in keyframe descriptions are instances in the 3D environment, and their poses should be constrained by relationships

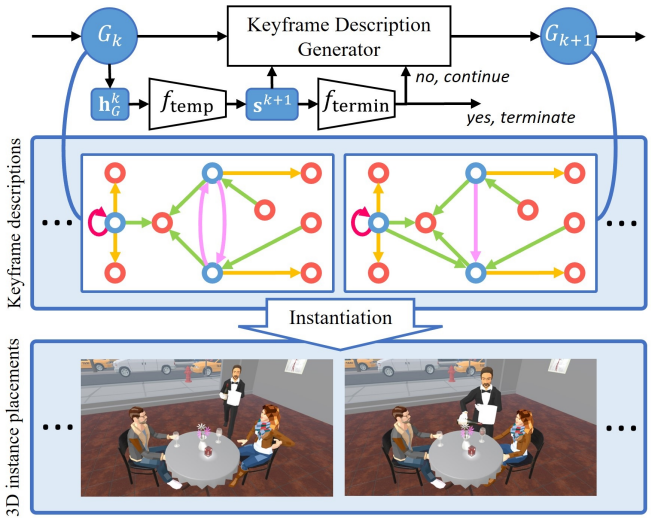

Fig. 2. An overview of our approach. A sequence of keyframe descriptions $\{G_k\}$ is generated by repeatedly producing new keyframe descriptions conditioned on their predecessors, considering spatial-temporal relations. A recurrent temporal function f_{temp}, an LSTM in our implementation, takes in the graph summary $\mathbf{h}_G^k$ and outputs a generation signal $\mathbf{s}^{k+1}$. A termination function f_{termin}, an MLP in our implementation, decides whether to end the iterative generation. In the next step, an activity snippet instantiation model, based on an optimization framework, synthesizes placements of instances encoded by the keyframe descriptions as a sequence of 3D scenes.

encoded by the edges throughout the interactions among instances. We use an optimization framework to refine the 3D instance placements subject to the constraints.

4 ACTIVITY SNIPPET REPRESENTATION

We represent an **activity snippet** as a sequence of keyframes together with their descriptions. In our work, a keyframe not only refers to a moment in the snippet, but also encodes 3D placements of instances (including characters and objects) at that moment. Formally, an activity snippet comprising T keyframes is defined as $\Gamma = \{(G_k, \Phi_k)\}_{k=1,2,\ldots,T}$, where G_k is the **keyframe description** of the k-th keyframe and is represented as a graph. Φ_k is a set of parametric 3D poses of instances in the scene of the k-th keyframe. As an illustrative example, Figure 3 shows some keyframe descriptions of an activity snippet.

4.1 Representing Frame Descriptions as Graphs

To describe abstract interaction relationships in a scenario with multiple characters and objects, we represent a keyframe description as graph $G_k =< V_k, E_k >$ containing a vertex set V_k and an edge set E_k. Graphs are general, informative, and flexible for modeling diverse interactive schemes. A set of node categories (encoding characters and objects) and edge categories (encoding interaction relations and self-statuses) are used for keyframe descriptions.

Node Features. The set of node categories is $\mathcal{N} = \{n\} = \mathcal{N}_{\text{ch}} \cup \mathcal{N}_{\text{obj}}$, where $\mathcal{N}_{\text{ch}}$ and $\mathcal{N}_{\text{obj}}$ denote the sets of character categories (blue nodes in Figure 3) and object categories (red nodes in Figure 3),

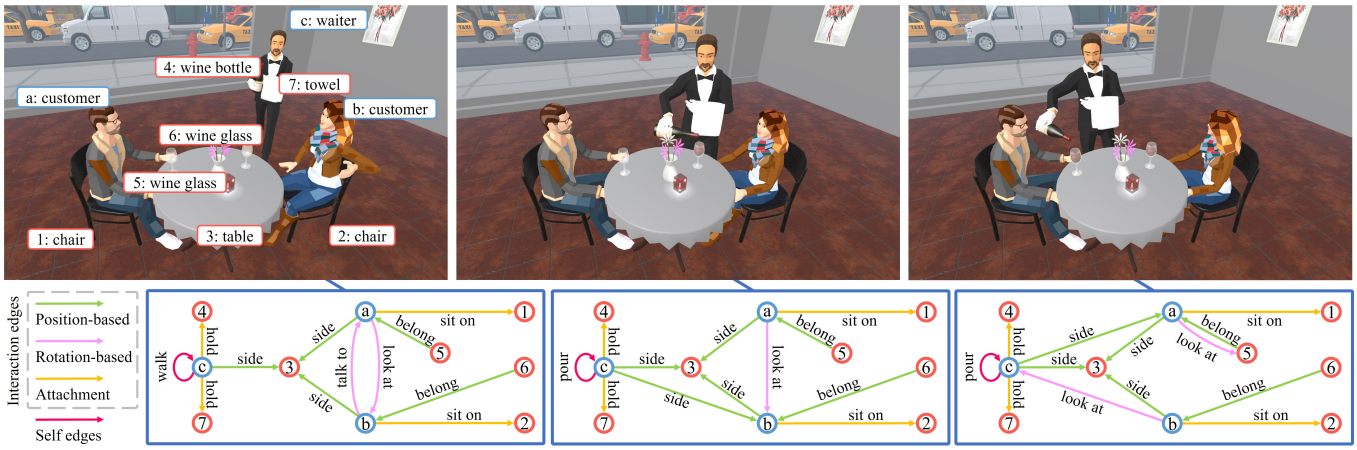

Fig. 3. A generated "serve wine" activity snippet. 3D placements of instances (characters and objects) and their animation assignments are guided by interactions encoded by edges in the keyframe descriptions. Blue nodes denote characters. Red nodes denote objects. Edge colors indicate edge categories.

respectively. A node v is associated with its category embedding vector $\mathbf{x}_v$.

Edge Features. We define the set of edge categories $\mathcal{E}_{\text{edge}} = \{e\} = \mathcal{E}_{\text{ia}} \cup \mathcal{E}_{\text{ria}} \cup \mathcal{E}_{\text{self}}$. An edge e is associated with its category embedding $\mathbf{x}_e$. $\mathcal{E}_{\text{ia}}$ stands for interaction edges, which may further be categorized into three sub-types (position-based, rotation-based, and attachment) according to the constraint types (Section 6.1) in our optimization framework, as shown in Figure 3. Note that while generic edge categories are defined as directed, we define their reverse categories to support undirected graphs for message-passing on graphs [Gilmer et al. 2017]. For example, if a node u has a directed *look at* edge (u, v) pointing to node v, we automatically add a symmetric edge (v, u) with a reverse feature, which could be interpreted as *being looked at*. Therefore, $\mathcal{E}_{\text{ia}}$ includes directed interaction edge categories, and $\mathcal{E}_{\text{ria}}$ includes the reverse categories, and a bijection function $f_{\text{ia}} : \mathcal{E}_{\text{ia}} \rightarrow \mathcal{E}_{\text{ria}}$ pairs the two sets. Besides, the edge categories in $\mathcal{E}_{\text{self}}$ describe possible self statuses (e.g., a character is *walking*), which can only be assigned to a self edge (v, v) that goes out from and into the same node v. Note that as multiple edges of different categories may exist between the same pair of nodes u and v (e.g., character u may be *on the side of* while *in contact with* an object v), we merge such edges as a single edge (u, v) and stack the edge category embeddings $\mathbf{X}_{(u,v)} = \{\mathbf{x}_e\}$ for every original edge e from node u to node v.

4.2 Parameterizing 3D Poses of Instances

In a scenario with multiple characters and objects, we let an *instance* refer to either a virtual character or an object. An object is attributed by (p_o, θ_o), where p_o is the position and θ_o is its orientation. Similarly, a character is attributed by $(p_c, \theta_{cb}, \theta_{ch})$, where p_c is the character's position, θ_{cb} is the body orientation, and θ_{ch} is the head orientation relative to θ_{cb} as shown in Figure 3. Therefore, the k-th keyframe's instances are parameterized as $\Phi_k = \big\{\{(p_c, \theta_{cb}, \theta_{ch})\}, \{(p_o, \theta_o)\}\big\}$.

5 GENERATING KEYFRAME DESCRIPTIONS

Our approach generates keyframe descriptions as graphs to depict a sequence of dynamic interaction scenarios. Particularly, a keyframe description G_k should be conditioned on its predecessor G_{k-1} in the sequence to maintain temporal relations. In addition, we also expect our generative model to be conditioned on particular activity labels, so that users may specify their desired activity types and our model can generate corresponding keyframe descriptions. For example, for a restaurant scene, a snippet could be generated to show the activity of "taking a food order" or "serving customers while they dine".

While the generated graphs should describe frames at discrete time points, they also should preserve some continuity such that changes between adjacent keyframe descriptions are gradual and smooth. We thus devise a generative model based on DGMG [Li et al. 2018], which is a sequential generative method. DGMG has been successfully applied in other graphics tasks like layout synthesis [Wang et al. 2019a]. To construct a graph sequentially, it makes the following decisions iteratively: (1) Should a new node be added to the graph? (2) If a new node is added, should an edge be added to connect this new node to an existing node? (3) If a new edge is added, which existing node in the graph should the edge be connected to?

This generative framework, however, is only able to continuously add nodes and edges to the graph. In order to support generating sequenced keyframe descriptions, we expect our generative model to be able to *delete* existing nodes and edges as well: if such general modifications including deletions in existing graphs are allowed, a keyframe description G_k can be generated by using G_{k-1} as a base and applying modifications. For example, if two characters chat in a keyframe, they may end the conversation (as indicated by deleting the edge representing the "talk" relationship), and a character may leave the scene (as indicated by deleting the node representing the character), at a subsequent keyframe.

The following section describes the basic background of message-passing graph convolution and details of the modification actions.

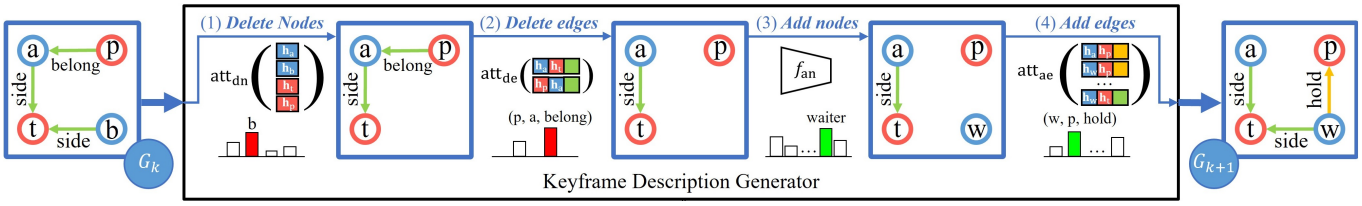

Fig. 4. The workflow of the keyframe description generator. Starting from an input keyframe description G_k, it triggers four modules sequentially to delete nodes, delete edges, add nodes, and add edges, and finally produces the next keyframe description G_{k+1} conditioned on G_k. In this example, nodes a and b refer to two customers. Nodes w, t, and p refer to a waiter, a table, and a plate, respectively.

5.1 Message Propagation on Graphs

During message propagation, each node receives messages from its neighbors. The received messages are aggregated and used to update the node feature. The aggregated message $\mathbf{a}_v$ for node v is:

$$\mathbf{a}_v = \sum_{(u,v)} f_{\text{msg}}(\mathbf{h}_u, \mathbf{h}_v, \mathbf{h}_{(u,v)}). \tag{1}$$

Here, f_{msg} computes the message vector from a neighbor u to node v. $\mathbf{h}_u$ and $\mathbf{h}_v$ are latent node features of node u and node v. Since multiple edges of different categories may exist from node u to node v, $\mathbf{h}_{(u,v)}$ is an aggregated edge feature given the stacked edge category embeddings $\mathbf{X}_{u,v}$:

$$\mathbf{h}_{(u,v)} = \text{aggr}_e(\mathbf{X}_{u,v}) = \sum_{\mathbf{x}_e \in \mathbf{X}_{u,v}} \sigma(g_e(\mathbf{x}_e)) \odot f_e(\mathbf{x}_e), \tag{2}$$

where g_e is a function that maps an edge embedding vector $\mathbf{x}_e$ to an "importance" value, activated by a logistic sigmoid function σ. f_e is an edge feature function. As a result, the aggregation function aggr_e outputs a weighted sum of edge features.

With the aggregated message $\mathbf{a}_v$ from neighbors and the current node feature $\mathbf{h}_v$, the updated feature $\mathbf{h}'_v$ is computed by a node feature function f_n such that $\mathbf{h}'_v = f_n(\mathbf{a}_v, \mathbf{h}_v)$.

5.2 Generating the Next Keyframe Description

In the generation process, picking one candidate out of a varying number of candidates is often needed (e.g., deleting a node, adding an edge between a pair of nodes). We use a scaled dot-product attention operator [Vaswani et al. 2017] for candidate selection. The probability $p_{\mathbf{k}}$ of selecting the candidate associated with key $\mathbf{k}$ is:

$$p_{\mathbf{k}} = \text{att}(\mathbf{q}, \mathbf{k}) = \text{softmax}_{\mathbf{k}}(\frac{1}{\sqrt{d}}(\mathbf{q}\mathbf{W}_{\mathbf{q}})(\mathbf{k}\mathbf{W}_{\mathbf{k}})^T), \tag{3}$$

where $\mathbf{W}_{\mathbf{q}}$ and $\mathbf{W}_{\mathbf{k}}$ are learnable matrices that project the query $\mathbf{q}$ and the key $\mathbf{k}$ into d-dimensional spaces. In our implementation, all modules that need the attention operator consider different keys but the same query $\mathbf{h}^*_G$, which is an augmented graph summary:

$$\mathbf{h}^*_G = [\mathbf{h}_G, \mathbf{s}], \text{where } \mathbf{h}_G = \text{aggr}_G(\{\mathbf{h}_v\}). \tag{4}$$

This definition is based on two factors that we want to include in order to guide the graph modification: (1) the original graph summary $\mathbf{h}_G$, which represents the up-to-date overall interaction relations encoded by the current graph G and is read out using an aggregation function aggr_G same as Equation 2 but with its own parameters; and (2) the generation signal $\mathbf{s}$ that encodes the temporal history of the graph sequence, of which details are discussed later in Section 5.3. Based on these notations, we introduce the following four modules, as depicted in Figure 4, to generate the next keyframe description:

(1) *Delete nodes.* A critical need in our task is to delete nodes. For example, a waiter may take away empty plates that appear in previous frames. All associated edges are also removed when deleting a node. The probability of deleting node v is $\text{att}_{\text{dn}}(\mathbf{h}^*_G, \mathbf{h}_v)$ considering the node feature $\mathbf{h}_v$ as the key.

(2) *Delete edges.* Akin to the *delete node* module, interactions may change in a sequence of frames thus our approach needs to be able to delete edges. In this module, the probability of deleting the edge category e in edge (u, v) considers the pairwise node features together with the edge category feature, and is computed as $\text{att}_{\text{de}}(\mathbf{h}^*_G, [\mathbf{h}_u, \mathbf{h}_v, \mathbf{x}_e])$. When computing such probabilities, reverse edge categories $e \in \mathcal{E}_{\text{ria}}$ are not considered. Instead, if a non-self-edge (u, v, e) is deleted, its reverse edge $(v, u, f_{\text{ia}}(e))$ is also deleted from the graph.

(3) *Add nodes.* Since the node category is the only factor to consider when trying to add a node, this module does not need an attention operator to handle a varying number of candidates. Instead, the probability of adding a node of a certain category is obtained from $\text{softmax}(f_{\text{an}}(\mathbf{h}^*_G))$, where f_{an} is a function that outputs logits associated with all node categories. Once a new node u is added, its node feature $\mathbf{h}_u$ is initialized through a function f_{init} such that $\mathbf{h}_u = f_{\text{init}}(\mathbf{x}_u, \mathbf{h}^*_G)$, where $\mathbf{x}_u$ is the category embedding of node u.

(4) *Add edges.* Considering all possible combinations of pairwise directed edges and edge categories, the probability of adding an edge of category e from node u to node v is computed as $\text{att}_{\text{ae}}(\mathbf{h}^*_G, [\mathbf{h}_u, \mathbf{h}_v, \mathbf{x}_e])$. Reverse edges are excluded. The edge category is restricted such that $e \in \mathcal{E}_{\text{self}}$ when trying to add a self-edge, and otherwise $e \in \mathcal{E}_{\text{ia}}$. Once a non-self-edge (u, v, e) is added, its reverse edge $(v, u, f_{\text{ia}}(e))$ is added simultaneously. Note that in case the number of keys (i.e. combinations of edge source, destination, and category) is too large, the attention operator's performance will likely decrease. A substitute method to pick candidate keys is to compute log probabilities for adding each possible edge-feature combination (v, u, e) through a function (e.g., MLP), concatenate them into one distribution, and sample from it, as suggested in [Wang et al. 2019a].

Overall, a single round of keyframe description generation triggers the modules following the order listed above. Each module runs repeatedly until it decides to stop, and then the process moves on to the next module. Figure 4 shows the overall workflow. This order follows the logic that unneeded nodes/edges should first be removed (since deleting nodes removes related edges, deletion goes first), before new content is added (new edges could be associated with new nodes, so it starts by adding nodes). In a module that uses the attention operator, an empty feature is used for calculating the probability of "STOP", which indicates that this module should terminate and the next module should start. In the *Add node* module that outputs logits of a fixed shape corresponding to node categories, an additional "STOP" category is included.

5.3 Preserving Temporal Relations in the Sequence

In order to generate a sequence of keyframe descriptions, a temporal module is applied to preserve temporal relations and to further guide the generative model to produce subsequent keyframe descriptions. We use the following temporal function f_{temp}:

$$\mathbf{s}^{k+1} = f_{\text{temp}}(f_{\text{t}}(\mathbf{h}_G^k, \mathbf{c}), \mathbf{s}^k), \tag{5}$$

where $\mathbf{c}$ is a highest-level condition vector that encodes the target activity, and the function f_{t} combines the graph summary $\mathbf{h}_G^k$ for keyframe k with $\mathbf{c}$. Note that at the initial keyframe, $\mathbf{s}^0$ is a random vector. During training and normal inference, the initial keyframe description G_0 is an empty graph and the generation process only needs to call the *add nodes* and *add edges* modules to generate G_1. However, G_0 can also be provided as a specific initialization or set as a partial graph during the inference if needed. We demonstrate a possible application scenario of this feature for experiencing activity snippets in mixed reality in Section 7.4.

Following this step, whether the graph sequence generation should continue is judged by another function $f_{\text{termin}}(\mathbf{s}^{k+1})$. If the generation does not terminate, $\mathbf{s}^{k+1}$ is a new generation signal vector used for guiding the next graph generation at frame $k+1$. As a result, a sequence of keyframe descriptions can be generated iteratively.

5.4 Dataset

We train the keyframe description generator using data from the MOMA [Luo et al. 2021] dataset. Annotations are represented using hypergraphs, which depict multi-object, multi-character activities and high-level interactions of crowded and complex scenes in video clips. For each video clip, a sequence of keyframes is picked and annotated along the timeline. MOMA uses different labels for node categories (e.g., character “waiter”, object “menu”), and edge categories (e.g., dynamic relationship “talk to”, static relationship “next to”). Clips are epitomized using activity labels (e.g., “serve food”). While those annotations are originally attributed with bounding boxes of characters and objects in the video, we only use the abstract graphs in our work, and thus exclude redundant graphs in a subsequence if they share the same structure. An additional modification of the data is that we add affiliation relationships to the original graphs if needed. For example, in a dining activity, if a waiter delivers a plate to a customer, by parsing the original graph we may only know that the plate is placed somewhere on the table, but which customer the plate is delivered to is unclear. In this example, we add an edge *belong* between the plate and its affiliated customer by checking the positions of their bounding boxes in the video.

To support the sequential generative model, orders of graph generation are needed. To complete the same target graph generation, one possible order may refer to adding some nodes first, while another order may prioritize adding some other nodes. For example, if a cup and a dish are to appear in the next keyframe, adding whichever first leads to the same new scene, and advancements of higher-level activities are unaffected. We chose random ordering in our implementation. In contrast to DGMG [Li et al. 2018] which only cares about the order of creating a single graph as the outcome, our model is expected to also continuously capture orders of changes between graphs in adjacent frames because we want to use the synthesized sequence of graphs for generating an activity snippet animation. To create a random order of transition from the keyframe description G_k to G_{k+1}, we compare and record the sets of deleted nodes, deleted edges, added nodes, and added edges, randomly shuffle all sets, and then merge them together to form a valid transition order.

5.5 Implementation Details

We used $T = 3$ rounds of message propagation. We used neural networks for functions during the graph generation. Specifically, we used (1) linear layer for functions g_e and f_e; (2) multilayer perceptron (MLP) for functions f_{msg}, f_{an}, f_{init}, f_t and f_{termin}; (3) gated recurrent unit (GRU) for the node feature function f_n; and (4) long short-term memory (LSTM) for the temporal function f_{temp}. The training of the sequential generative model was consistent with the original DGMG [Li et al. 2018].

6 INSTANTIATING ACTIVITY SNIPPETS IN 3D SCENES

Given generated keyframe descriptions that only encode abstract interactions among instances (characters and objects), the following challenge is to instantiate activity snippets by posing and animating characters and objects accordingly, considering spatial relations in each keyframe and temporal relations throughout the whole activity snippet. We propose an optimization-based approach, which considers pairwise node constraints converted from edges in keyframe descriptions, to synthesize plausible 3D placements of instances at the keyframes. Pre-defined symbolic animations are later assigned to instances with optimized poses, and intermediate animation states are interpolated during transitions between keyframes.

6.1 Converting Edges into Pairwise Constraints

We model interactions encoded by the edges of keyframe descriptions as soft constraints, and formulate such constraints as cost functions used for optimization. Each cost function $E(u, v)$ measures the quality of posing a pair of nodes u and v to satisfy the corresponding edge category. Note that self-edges defined in our work depict nodes’ statuses. Thus they are used for creating symbolic animations without considering instances’ positions and orientations, and they are not incorporated into the optimization. Note also that reverse edges are only used for message passing during the keyframe description generation in Section 5, and we only consider

directed edges during the activity snippet instantiation. Generally, we categorize the constraints into three types:

6.1.1 Position-based Constraints. Denoted by $\{C_\text{p}\}$, these constraints encode spatial relations between nodes. For example, if u is *on the side of* v, u should be close enough to v such that the distance between them is shorter than a threshold. The general definition of a position-based constraint between instances u and v is:

$$C_\text{p}(u,v) = \max\left(1 - e^{\lambda D - d(u,v)}, 0\right) \tag{6}$$

where $d(u,v)$ is a distance function between u and v, which can be flexibly defined for specific constraints and the applied environments. D is a specified target distance. $\lambda \geq 1$ is a relaxation factor that allows temporarily tolerating a larger target distance and is controlled by the optimization solver described in Section 6.2.1.

We describe details of the distance function $d(u,v)$ and target distance D for some most common position-based constraints. We include some additional definitions in our supplementary material.

- *On the side of.* Though the specification of D often depends on the geometries, we set $D = 0.4$m by default. If u and v are both characters or item objects (i.e. movable and graspable), the distance is easily defined as $d(u,v) = |p_u - p_v|$. However, a more common case in our work is that a character u is *on the side of* a furniture object v. In this case, a nearest point p^*_{uv} is found on the boundary of v, and the distance is $d(u,v) = |p_u - p^*_v|$, as indicated by the yellow line segment in Figure 5. In some special cases, characters' roles and objects' functionalities would bring out additional considerations for defining the distance. For example, in the "front desk reception" example in Figure 8, when computing the customer's and the waiter's distances to the desk, the nearest point p^*_v is only calculated on the boundary at the desk's front surface. On the contrary, the nearest point is found at the desk's back surface for the receptionist.
- *In contact with.* If u is in contact with v, v should ideally be reachable from u and we set target distance $D = 0.3$m by default. We also apply the same setting for the edge *belong*, thus if an object belongs to a character, the object is expected to be reachable. The distance is usually computed as $d(u,v) = |p_u - p_v|$. After a character puts or before the character picks up an item at some keyframes, an implicit *in contact with* constraint is added such that the character should be posed close to where the item is placed. Variations are allowed if u is expected to contact a specific region of v. For example, in the "barber service" example in Figure 8, barbers may only contact the customers' heads, thus the distance is defined as $d(u,v) = |p_u - p^\text{head}_v|$, where p^head_v is v's head position.

6.1.2 Orientation-based Constraints. Denoted by $\{C_\text{o}\}$, these constraints are about facing directions. For example, if u *looks at* v, u should face towards v. The general definition of an orientation-based constraint between instances u and v is:

$$C_\text{o}(u,v) = \frac{1 - s(u,v)}{2}, \tag{7}$$

where $s(u,v)$ is an orientation suitability function between u and v. $s(u,v)$ is generally a dot product between two orientation unit vectors. Specific definitions about what orientation vectors are associated with u and v may vary with the constraints and scenes.

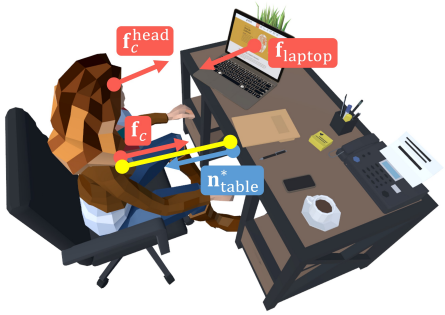

Fig. 5. An example scene where a character is sitting *on the side of* a table and is *looking at* a laptop. Red vectors $\mathbf{f}_c^\text{head}$, $\mathbf{f}_c$ and $\mathbf{f}_\text{laptop}$ denote the forward direction of the character's head, character's body, and the laptop, respectively. The yellow line segment indicates the distance between the character at position p_c and the nearest position p^*_table on the boundary of the table. $\mathbf{n}^*_\text{table}$ denotes the normal vector at position p^*_table.

When considering the orientation of an instance, we denote the instance's forward direction as a unit vector $\mathbf{f}$. If the instance is a character, since we consider a separate orientation parameter for the head, we also denote the facing direction as $\mathbf{f}^\text{head}$. We provide definitions of the orientation suitability function $s(u,v)$ for some common orientation-based constraints, as follows:

- *Look at.* When a character u looks at v, we expect that u faces v directly. Therefore, we encourage u's facing direction $\mathbf{f}^\text{head}_u$ to align with the vector $\mathbf{v}_{uv} = \frac{p_v - p_u}{|p_v - p_u|}$ from u to v, thus $s(u,v) = \langle \mathbf{f}^\text{head}_u, \mathbf{v}_{uv} \rangle$. In case v is an object with certain functionalities (e.g., a character *looks at* a laptop as shown in Figure 5), an auxiliary orientation-based constraint is simultaneously applied with $s(v,u) = \langle \mathbf{f}_v, \mathbf{v}_{vu} \rangle$ to constrain object v to face u as well. The same definition is also set for the constraint *talk to*.
- *On the side of.* While a position-based constraint is defined for this edge label, we additionally constrain instance u's orientation when it is *on the side of* a furniture object v. Recall that the nearest point p^*_v is found at the boundary of v, we denote the normal vector at p^*_v as $\mathbf{n}^*_v$, as illustrated by the blue vector in Figure 5 which shows a character facing a table. To encourage u's body direction $\mathbf{f}_u$ to align with the inverse of $\mathbf{n}^*_v$, we set $s(u,v) = \langle \mathbf{f}_u, -\mathbf{n}^*_v \rangle$.
- To create more plausible activity snippets, other intuitive considerations are also encoded as orientation-based constraints though they do not appear in the keyframe descriptions. For example, if a character u *sits on* a chair v such that an attachment relation exists, u's and v's forward directions should align with each other, and thus $s(u,v) = \langle \mathbf{f}_u, \mathbf{f}_v \rangle$ is used.

6.1.3 Attachment Constraints. Most of these are hard constraints applied to enforce that the attached child instance's position is always consistent with its parent instance throughout the optimization. Common examples include *sit on*, *hold*, etc. In practice, we combine such attachment hierarchies as single instances throughout the

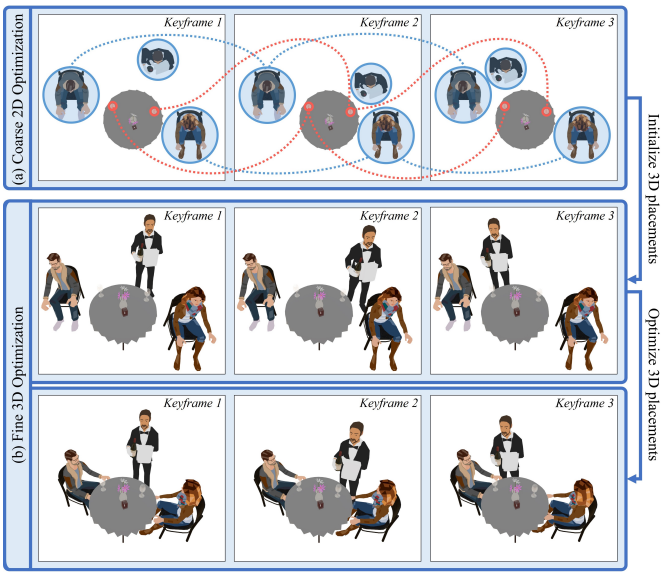

Fig. 6. Our approach instantiates activity snippets in 3D environments by a two-stage optimization. (a) A coarse optimization is performed in 2D space, whose results are projected back to the 3D environment as the initialization for (b) a fine optimization in 3D space.

coarse 2D optimization (Section 6.2.1). In the fine 3D optimization (Section 6.2.2), the attached instance is free to change its orientation. An exception of enforcing a child's position to be consistent with its parent's position is *above*, which allows the child object to move freely on the top surface of the parent furniture object.

6.2 Optimizing Instance Placements

In preliminary tests, we observed that optimizing the placements of all instances in a whole activity snippet using the generated keyframe description sequence is challenging, considering the tangled spatial constraints within frames and cross-frame temporal constraints. In particular, since the graph dimensionalities (i.e. number of nodes and edges) of keyframe descriptions are variable, the optimization becomes more difficult as the number of edge-based constraints increases.

We therefore devise a two-stage optimization process as shown in Figure 6: (1) In a coarse 2D optimization, we first project the 3D scene onto a 2D space, and optimize the 2D placements by only considering position-based constraints. Starting from a random initialization, we first relax all applied constraints and progressively tighten them throughout the optimization process. (2) In a fine 3D optimization, we retrieve the 3D poses of instances from the 2D optimization result and use them as the initialization for another round of 3D placement optimization.

In both stages, we employ a consistent optimization solver: we use simulated annealing [Kirkpatrick et al. 1983] with a Metropolis-Hastings state-search step [Hastings 1970; Metropolis et al. 1953] to explore the complex spatial-temporal solution space. Given an activity snippet Γ, a Boltzmann-like objective function is defined as:

$$f(\Gamma) = e^{-\frac{1}{t}C(\Gamma)}, \tag{8}$$

where $C(\Gamma)$ denotes the constraints considered in the activity snippet Γ. t is the temperature parameter that drops over iterations by a decay factor until it reaches a low value near zero. In each iteration, based on the current activity snippet Γ, a tentative instantiation Γ' is proposed, which is accepted with a probability:

$$\alpha(\Gamma'|\Gamma) = \min\left[\frac{f(\Gamma')}{f(\Gamma)}, 1\right] = \min\left[e^{\frac{1}{t}(C(\Gamma)-C(\Gamma'))}, 1\right]. \tag{9}$$

In practice, we fix a furniture object (e.g., a table) as the center and then optimize other instances' poses around the center object. We discuss the details of the two optimization stages in the following.

6.2.1 Coarse Optimization in 2D Space. The goal of this stage is to quickly generate a coarse but reasonable initialization for the fine optimization in 3D space. Our observation is that position-based constraints are generally harder to satisfy throughout the optimization compared to orientation-based constraints, because coping with multiple position-based constraints in different frames associated with the same instance is common yet challenging. Suppose two customers sit somewhere near a table and later on a waiter comes to serve them successively. In this example, the waiter's later movements depend on where the two customers sit, hence the waiter's positions in the later frames should be optimized considering the customers' positions in the earlier frame, rather than just considering the local instance distributions in the later frames.

With this consideration in mind, we project instances onto a 2D space as shown in Figure 6 (a), and use bounding circles to represent them. Note that hierarchies and attachments are preserved: if a character sits on a chair, these two instances are merged using a single bounding circle. On the other hand, if a wine glass is put on the table, the table's top surface is its valid moving area and 2D collisions between them are ignored. In case movable objects appear in the scene but are not associated with any characters or furniture objects, we implicitly bind them with furniture containing top surfaces. Those movable objects may jump among multiple top surfaces during the optimization, and their final positions (i.e. staying at where of which surface) may depend on other constraints. For example, the bag in the "front desk reception" example in Figure 8 is placed on the upper surface of the desk but not the lower surface near the receptionist, because the bag should be accessible to the customer. We expect that, after the optimization, all instances move to reasonable positions such that position-based constraints are roughly satisfied.

Even when the scenario is simplified into 2D, optimizing the placements from a random initialization is still difficult especially when the temporal sequence is long. We provide a preliminary study in our supplementary material, which shows that when dealing with a long list of constraints, standard simulated annealing may only satisfy a small portion of them and get stuck at local minimum.

To this end, we propose a progressive optimization with a varying relaxation degree, which splits the conventional optimization process into different sub-optimization phases, and gradually tightens the constraints. The control of relaxation is achieved by adjusting

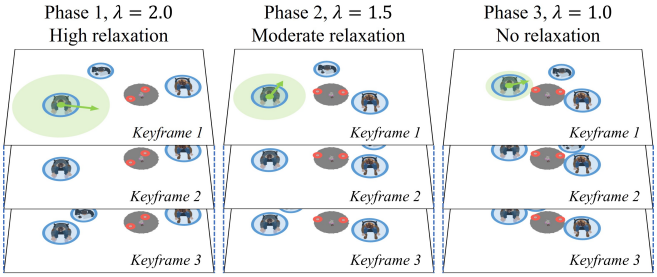

Fig. 7. An example progressive optimization with three sub-optimization phases, where the relaxation factor λ drops from 2 to 1. The final placements of instances in all phases are shown in the columns. The larger λ is, the more target distances are relaxed in the position-based constraints, and the larger move ranges are allowed as depicted by the green circles. Instances' possible moves during the optimization, as shown by the green arrows, are sampled within the move ranges.

the factor λ in Equation 6. Phases are defined along with the temperature decrease (i.e. as the temperature drops below the next threshold, the optimization transitions to the next phase). Figure 7 shows an example. The initial temperature of a sub-optimization phase equals the final temperature of the previous phase, such that the greediness of accepting samples of lower costs increases after the phase transition.

In the initial phase, the constraints are most relaxed to encourage quick convergence to a local minimum. Meanwhile, random moves are sampled using the largest range to allow fast jumps within the energy landscape. Every time a transition to the next phase occurs, the constraints are tightened a little bit by decreasing the relaxation factor λ until it equals 1. Essentially, when transitioning to a new phase, the optimization objective changes and the solver is transferred to a stricter energy landscape, where the following optimization is triggered using the optimization result of the previous phase as the initialization. In our implementation, we set the decrease of the relaxation factor λ from 2 to 1 evenly among the phase transitions by default. The number of phases is typically 2 or 3, but could be flexibly set according to the optimization difficulty, which depends on the length of the activity snippet and the number of constraints available.

Since we ignore orientation-based constraints in this stage, we simplify the solution space by only considering characters' positions $\{p_c\}$ and objects' positions $\{p_o\}$. During the optimization, a proposed move is made by randomly picking one instance, and proposing to apply a position change $p \rightarrow p + \lambda\delta p$, as depicted by the green arrows in Figure 7. Therefore, moves of larger magnitudes are proposed in early phases when the relaxation factor λ is larger.

The total cost at this stage is summarized as a weighted sum $C = \sum w_p C_p$, where w_p is the weight (set as 1 by default) associated with each position-based constraint C_p. The instantiation is optimized with the solver as discussed at the beginning of Section 6.2.

Note that if an instance's interactions with the surroundings are consistent in a few continuous keyframes, we enforce cross-frame moves for any proposed move applied to that instance. Examples are highlighted in Figure 6: the cross-frame positions of some instances are linked by dashed lines, thus an instance's movements induced by a proposed move are enforced to be consistent across frames. For example, if a character is translated in frame 2, that character should also be translated by the same extent in frame 1 and frame 3. Moreover, we only compute constraints associated with this instance once in those corresponding frames to save computations. This strategy is also applied in the 3D optimization stage later.

6.2.2 Fine Optimization in 3D Space. Given the initial placements produced by the 2D optimization, another finer optimization is triggered in the 3D space as shown in Figure 6 (b). More precise colliders with respect to the geometries are used to create collision-free activity snippets. At this stage, while we assume that instances in different frames have already been reasonably positioned to satisfy the position-based constraints, tunings of their positions are still necessary due to the differences in collider representations. An example is that a person's legs can be accommodated under a table's top surface in 3D space when sitting near the table, while such a scenario would have caused overlapping of their bounding circles during the 2D optimization.

At this stage, we consider both the position and orientation-based constraints, and thus optimize all parameters $\{(p_c, \theta_{cb}, \theta_{ch})\}$ of the characters and all parameters $\{(p_o, \theta_o)\}$ of the objects. During the optimization, a proposed move is made by randomly picking one instance, and proposing to apply either a position change $p \rightarrow p+\delta p$ or a rotation change $\theta \rightarrow \theta + \delta\theta$, with probabilities 0.2 and 0.8, respectively. This is to reduce the number of positional updates since positions are already near-optimal given the 2D optimization results. Particularly, if the picked instance is a character, it can either refer to a body rotation with a probability of 0.7 or a head rotation with a probability of 0.3 in our implementation.

The total cost at this stage is $C = \sum w_p C_p + \sum w_o C_o$, where w_p and w_o are weights (set as 1 by default) for position-based constraints and orientation-based constraints, respectively. The instantiation is also optimized with the solver.

7 RESULTS

7.1 Generating and Instantiating Activity Snippets

Figure 8 shows the selected keyframes of four generated activity snippets conditioned on different activity labels. Our supplementary material includes four additional results. Our video includes full animation sequences of all activity snippets.

Overall, the generated keyframe description sequences contain reasonable abstractions of multi-character, multi-object interactions: (1) Regarding spatial relations, the graphs are understandable and contexts in each keyframe are centered around the input activity labels. Relevant characters and objects (e.g., waitress and plates in the "serve food", receptionist in the "front desk reception") together with their interactions (e.g., applying gel in the "barber service", raising hands in the "presentation") are generated. (2) Regarding temporal relations, changes in descriptions during the keyframe transitions are well captured such that the activity advances in a reasonable order. For example, for the "serve food" activity, the waitress observes the customers' needs and delivers food to them one by one. For the "front desk reception" activity, the waiter guides the customer to the receptionist and then leaves.

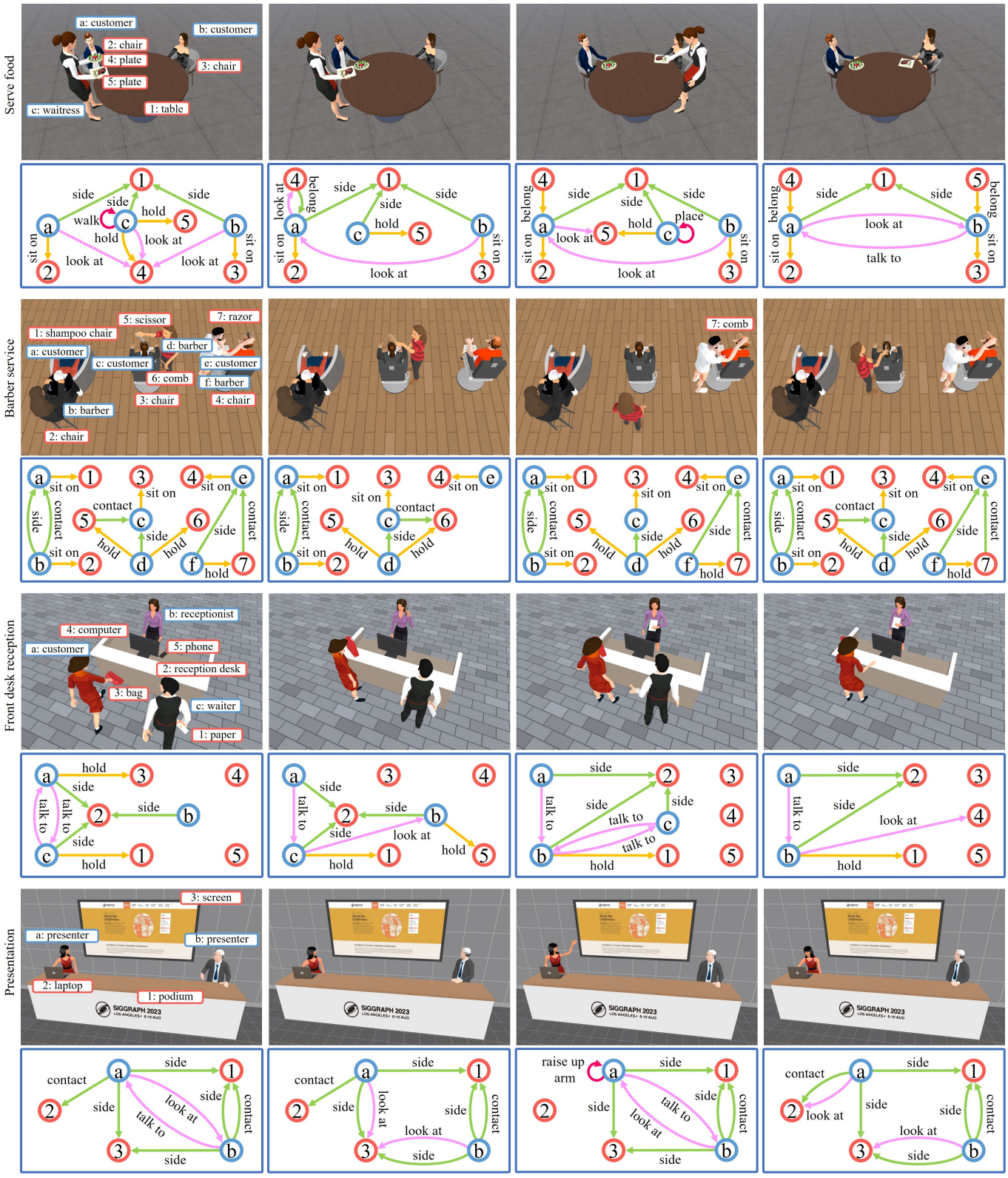

Fig. 8. Results of four generated activity snippets. Selected keyframes are presented.

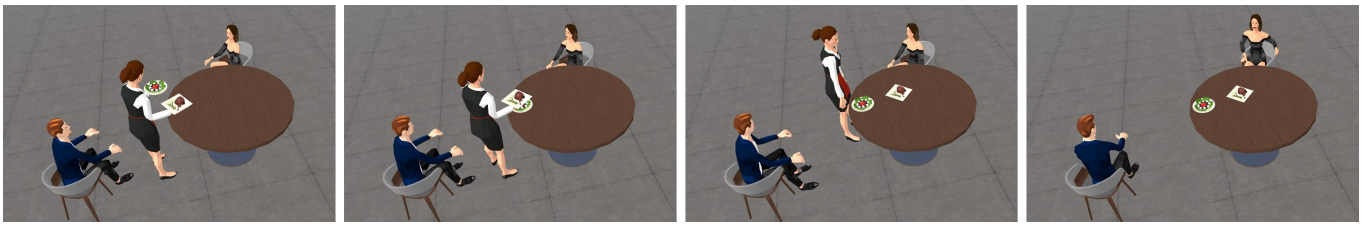

Fig. 9. Constraints are not reasonably shaped when a one-stage optimization is used to instantiate the same "serve food" activity as in Figure 8.

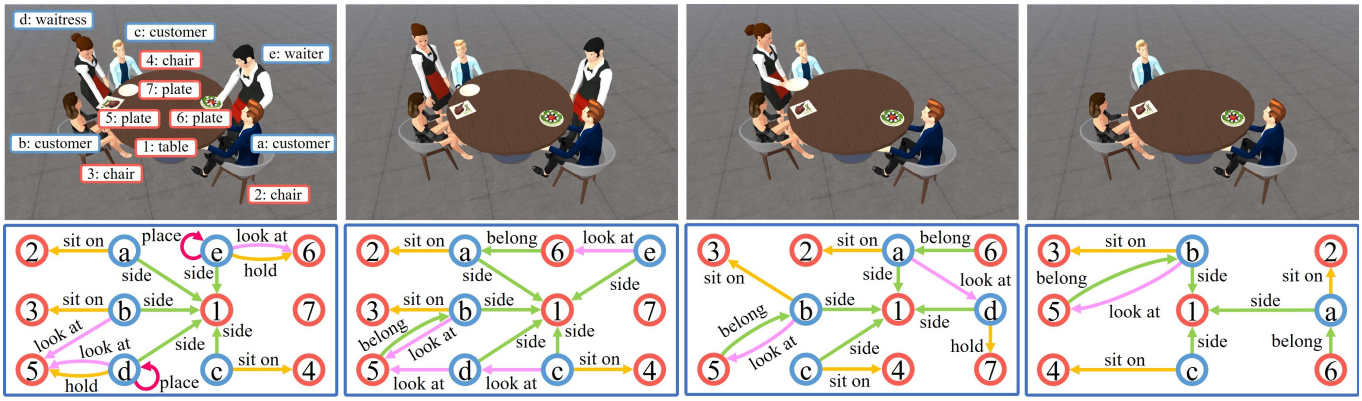

Fig. 10. Another generated activity snippet for the same "serve food" activity label as in Figure 8. Selected keyframes are presented.

Our optimization framework naturally poses the instances to enhance the quality of the 3D instantiations. Guided by the constraints converted from edges, the instances keep reasonable distances from each other, and interactions shaped by their facing directions also match the context. In complicated cases where some instances' placements are associated with constraints in different frames (e.g., in the "serve food" activity, where the waitress goes to deliver food depends on how the customers are posed), our approach finds optimal solutions to accommodate such constraints simultaneously. For comparison, we use a one-stage optimization that jointly optimizes all position-based and orientation-based constraints in the 3D space to create an ablation instantiation in Figure 9, which is of the same "serve food" activity as in Figure 8. The result suggests that it is difficult to find optimal activity snippet instantiations at one time, considering complex spatial-temporal relationships.

Note that different snippets can be generated for a given activity label. We show another generated "serve food" activity snippet in Figure 10 in addition to the one in Figure 8.

7.2 Learning from Synthetic Activities

When instantiating an activity snippet in a 3D environment, more details about the characters and objects in the scene and their interactions would help shape the placements. Since our regular training data in the MOMA dataset [Luo et al. 2021] is derived from 2D video clips, critical details about the scene could be missing due to occlusions or limited fields of view. 3D virtual environments, such as multiplayer games or the metaverse, can be ideal sources for collecting comprehensive 3D interaction data to better support the training of our method.

We conducted an experiment to demonstrate that our approach can learn from synthetic data created with either general or specific goals. While it was expensive to collect gaming data by ourselves, we synthesized training data via a simulation based on the rules and recipes of the popular multiplayer video game *Overcooked* [Games 2016]. Refer to our supplementary material for data preparation details. We synthesized data from the *Overcooked*'s setting because kitchens are common and highly functional workspaces: Tasks of preparing different dishes can typically be represented as a long sequence of sub-tasks, thus the definitions of cooking activities align with our problem representation. Our goal is to learn from synthetic cooking activities and generate realistic activity snippets about making dishes.

A generated "prepare burger" activity snippet is shown in Figure 11. In this result, one chef handles the cheese and another chef cooks the beef, and the generation consists of 19 keyframes. We compare this with training samples of the same length of keyframes. The mean graph edit distance (GED) from this generation to the closest training sample is 0.32. The results suggest that our keyframe description generation is capable of learning the dynamics behind such task-oriented activities (e.g., the beef must be chopped before being cooked in a pan).

7.3 Interactive Activity Snippets Generation

Our approach can be extended to engage a human player in an activity snippet interactively. The extension involves tracking the

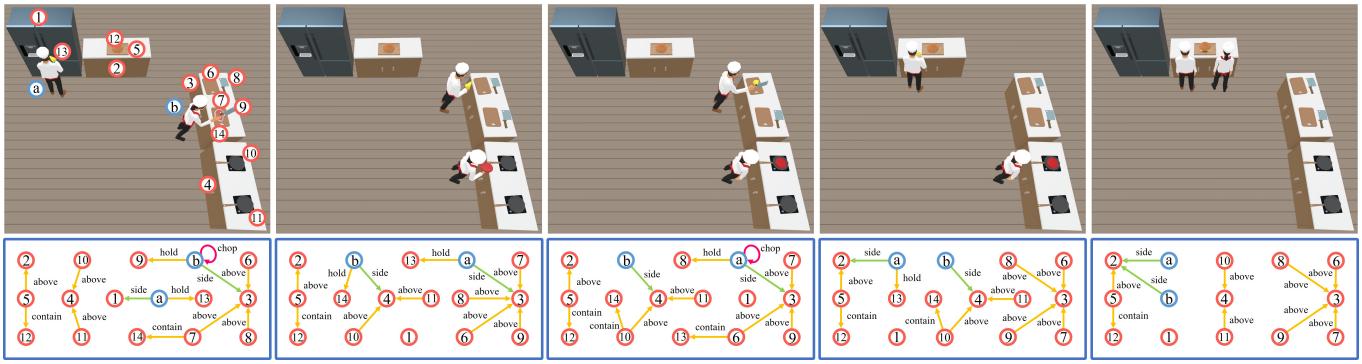

Fig. 11. A generated "prepare burger" activity snippet. The keyframe description generator was trained using the synthetic overcooked dataset. In the selected keyframes, chef *a* processes a block of cheese (object 13) and chef *b* processes a slice of beef (object 14).

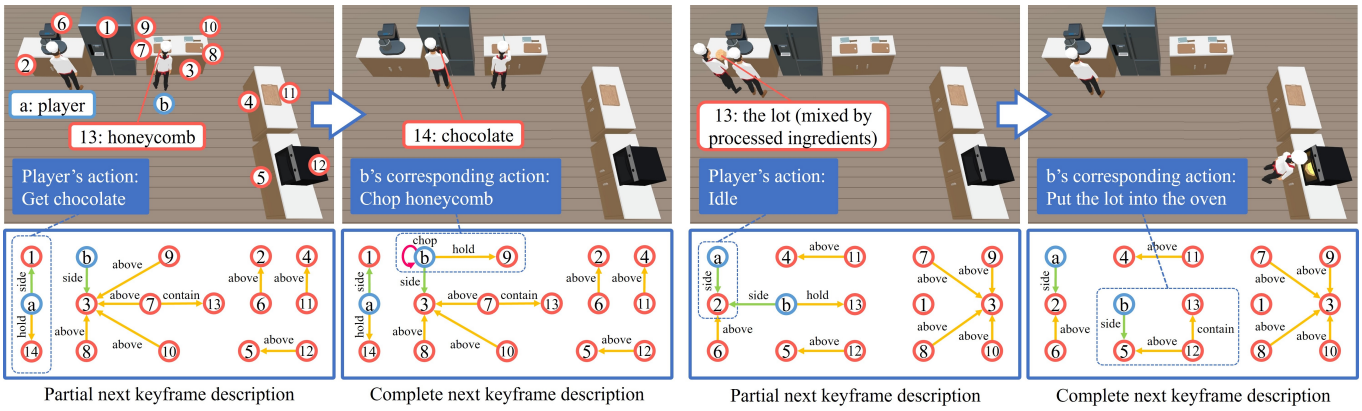

Fig. 12. An interactive "prepare cake" activity snippet. At each keyframe, the player makes an action to propose a local change, which leads to a partial next keyframe description based on the previous scene state. Following that, our approach generates the next keyframe description by making additional changes to the other character and objects in the scene.

player's behaviors, dynamically generating next keyframe descriptions, and instantiating them repeatedly. Specifically, at each keyframe, our approach advances the activity to the next keyframe as follows:

(1) Wait until the player has taken an action, update the corresponding changes onto the current keyframe description, and create a partial next keyframe description. Note that to conform to the sequential graph generation of our approach, such changes made by the player (not only the character the player controls, but also the objects involved in the interaction) are also converted into random graph transition orders through the same procedure discussed in Section 5.4.
(2) Conditioned on the partial next keyframe description, our approach proposes additional changes until the generation terminates, thus a complete next keyframe description is generated. During this step, changes in step (1) are fixed.
(3) Update 3D placements of characters (excluding the character controlled by the player) and objects in the scene by applying our optimization approach to instantiate the generated next keyframe description.

We also used the synthetic *Overcooked* activities in this experiment to create collaborative activities snippets. Figure 12 shows an example of a "prepare cake" activity snippet. To illustrate this feature, we developed a simple interactive interface that enabled a player to use the keyboard to move the character and the mouse to make symbolic actions (e.g., get chocolate, chop honeycomb). We also measure the mean GED to the closest training sample for this 21-keyframes interaction, and the result is 0.48. Theoretically, controls involving more degrees of freedom (e.g., body pose and hand pose controls to allow lower-level interactions such as object manipulation) can also be incorporated as long as the player's behaviors can be tracked and modeled in the same graph representation as in our approach.

7.4 Activity Snippets in Mixed Reality

Our generated activity snippets could be experienced in mixed reality (MR). In contrast to fully virtual scenarios, mixed reality needs to cope with both real and virtual instances concurrently. Our approach can adapt to specific physical environments by using real-scene initialization. To achieve this goal, the real scene graph should first be

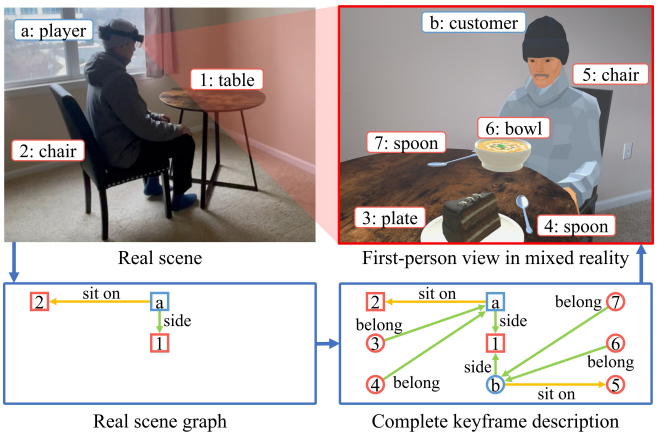

Fig. 13. A "serve food" activity snippet in mixed reality. The real scene graph was used as a partial keyframe description. The generated complete keyframe description, which included both real instances (indicated by rectangular nodes) and augmented virtual instances (indicated by circular nodes), was instantiated and visualized through a mixed reality headset.

extracted from the physical environment as depicted in Figure 13. We employed the settings as in [Li et al. 2022] to scan the 3D real environment and extract its geometric and semantic information. In our experiment, we tracked the head position and orientation of the player using a mixed reality headset (a Microsoft HoloLens 2), and we inferred that the player sat on the chair when the head coordinates projected onto the 2D floor plane were close to the chair's and when the height of the head decreased. If more tracking devices or sensors (e.g., hand-tracking gloves) are available, our approach could be extended to incorporate more types of interactions with mixed reality environments.

The real scene graph was used as a "partial graph" of the initial keyframe description that only encoded real characters and objects. Additional virtual nodes and edges were augmented using the keyframe description generator. During the keyframe description generation, the generator was refrained from modifying nodes and edges corresponding to real people and objects (i.e. setting zero probabilities of taking operations on them). The instantiation module then applied the optimization process to pose the virtual instances, and the instantiated activity snippet was presented through a mixed reality device.

7.5 Perceptual Study

We conducted a perceptual study to evaluate the quality of the activity snippets, considering both the generated keyframe descriptions and their instantiations. We recruited 300 participants on Amazon Mechanical Turk to rate 10 activity snippets in this study, including the 6 pieces shown in Figure 1, 3, and 8, and the additional 4 results in Figure 4 in our supplementary material.

We expected participants to rate the generations conditioned on real-world video recordings: For each trial of evaluating a generated activity snippet, we picked 5 video clips that have the same activity label and meanwhile the closest initial keyframe (i.e. counting the number and categories of nodes and edges, such that the videos

Table 1. Statistical results of participants' ratings on the reasonableness, intuitiveness, placement plausibility (Placement P.) and overall plausibility (Overall P.) of the generated activity snippets. The mean, median and standard deviation (SD) values are reported. Each metric is computed using 3,000 ratings for 10 activity snippets based on a 5-point Likert scale (1 meaning "the lowest" and 5 meaning "the highest").

	Reasonable	**Intuitive**	**Placement P.**	**Overall P.**
Mean	3.89	3.77	3.89	3.88
Median	4	4	4	4
SD	0.92	0.90	0.88	0.90

presented similar initial scene setups) from the MOMA [Luo et al. 2021] dataset, and showed these video clips before showing our generated results. The purpose was to let participants know about the original activity patterns of our learned model, which might help them evaluate if the scenes and interactions in our generated results generally match the observed patterns given the activity labels. We added short pauses when the play of activity snippets came to the keyframes. After showing both the example video clips and the activity snippet in each trial, we asked participants to rate on four metrics:

(1) Reasonableness: How naturally and realistically the activity advanced in general given the activity label, considering characters' interactions with each other and with objects.
(2) Intuitiveness: How well participants could understand what was happening in the scene.
(3) Placement plausibility: How plausibly instances were posed at the discrete keyframes.
(4) Overall plausibility: How plausibly the instances behaved overall considering both the poses and animations.

We use a 5-point Likert scale for all questions, with 1 meaning "the lowest" and 5 meaning "the highest". Metrics (1) and (2) were related to the quality of generated keyframe descriptions, while (3) and (4) were related to the instantiations and animations. We group the ratings by the metrics for all the 10 activity snippets and show statistical results in Table 1 with 3,000 ratings for each metric. Our supplementary material includes statistical results of ratings on each activity snippet. A majority chose 4 for all the metrics. The positive ratings suggest that, conditioned on observations from real-world recordings, the participants thought that our generated results embody the expected activity patterns in general, and the instantiations and animations reasonably recover the relationships encoded in the abstract descriptions.

8 LIMITATIONS AND FUTURE WORK

In this work, we present an approach to address a novel problem of activity snippet generation. We use a deep graph generative model to generate high-level keyframe descriptions for an activity snippet as a sequence of graphs, and instantiate those keyframes in virtual scenes through a two-stage optimization. Our experiment results validated the effectiveness of our approach and its flexibility to be applied in different application scenarios.

A typical failure case of our approach is that, when generating keyframe descriptions, nodes are sometimes added with no causality.

However, we expect that nodes, especially objects, should be added into the scene following some logic. For example, movable objects should be brought into the scene by characters, or be created on the condition that some other required objects exist and some specific actions are taken, such as the cooking examples discussed in Section 7.2. This issue is partly caused by the limitation of our training data: when annotating interactions captured in 2D videos, some objects that existed in a scene could be out of view from the camera at the beginning, but they might show up later as the camera moved. We believe that 3D interactions recorded in virtual environments, such as the metaverse or VR games, could serve as more comprehensive training data to mitigate this problem. We conducted an investigation in Section 7.2 to learn from synthetic activities. Another possible extension is to incorporate a more powerful temporal module (e.g. Transformer [Vaswani et al. 2017]) instead of a simple LSTM as in our implementation to learn the activity advancements.

While the training data in our work is derived from public videos, the use of real human activity data, especially from future 3D metaverse applications, may arouse privacy and ethics concerns. It is worth investigating important problems about preserving users' privacy (e.g., recordings of metaverse players' social behaviors) and resolving potential biases (e.g. gender, age groups) in the data.

Our current approach generates discrete keyframe descriptions of activity snippets. In the future, it is worth investigating defining edge labels of high granularity, such that minor changes in the scene can be encoded in the graph. Based on that, it is feasible to more densely sample keyframes and generate smoother interactions.

Our approach first generates abstract keyframe descriptions and then instantiates them in 3D scenes. As pose tracking and 3D interaction data become increasingly available in the future, precise poses of characters and objects can be recorded along with the high-level descriptions of interactions, such that an end-to-end model could be trained to generate activity snippets in one step. A related topic is to integrate motion synthesis techniques [2021a; 2019; 2021a] with our approach to substitute the interpolated transitive animations used in this work.

A compelling application in the future is to utilize our approach to drive an interactive authoring tool. Users may generate activity snippets either in a fully-automatic manner by only specifying activity labels, or in a semi-automatic manner by including manual authoring content or desired edits as partial constraints (like the MR experiment in Section 7.4) and driving our approach to complete the missing details.

An interesting topic is to apply our approach for AR/VR training or instructions [Chidambaram et al. 2021; Ipsita et al. 2022]. While we have validated that our approach could be extended to learn novel activities or tasks (Section 7.2), to support users' interactions (Section 7.3) and to facilitate mixed reality scenarios (Section 7.4), it is worth investigating collecting interaction sequences to complete specific tasks, using our approach to learn such dynamics, and employing the learned model to dynamically guide trainees step by step to finish the tasks based on their interactions with the scene.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their constructive comments. This project was supported by NSF grants (award numbers: 1942531 and 2128867).

REFERENCES

Giuseppe Abrami, Alexander Henlein, Attila Kett, and Alexander Mehler. 2020. Text2scenevr: Generating hypertexts with vannotator as a pre-processing step for text2scene systems. In *Proceedings of the 31st ACM Conference on Hypertext and Social Media*. 177–186.

Shailen Agrawal and Michiel van de Panne. 2016. Task-based locomotion. *ACM Transactions on Graphics* 35, 4 (2016), 1–11.

Nikos Athanasiou, Mathis Petrovich, Michael J Black, and Gül Varol. 2022. TEACH: Temporal Action Composition for 3D Humans. *arXiv preprint arXiv:2209.04066* (2022).

Yunfei Bai, Kristin Siu, and C Karen Liu. 2012. Synthesis of concurrent object manipulation tasks. *ACM Transactions on Graphics* 31, 6 (2012), 1–9.

Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. 2015. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the ieee conference on computer vision and pattern recognition*. 961–970.

Zhe Cao, Hang Gao, Karttikeya Mangalam, Qi-Zhi Cai, Minh Vo, and Jitendra Malik. 2020. Long-term human motion prediction with scene context. In *European Conference on Computer Vision*. Springer, 387–404.

Subramanian Chidambaram, Hank Huang, Fengming He, Xun Qian, Ana M Villanueva, Thomas S Redick, Wolfgang Stuerzlinger, and Karthik Ramani. 2021. Processar: An augmented reality-based tool to create in-situ procedural 2d/3d ar instructions. In *Designing Interactive Systems Conference 2021*. 234–249.

Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. 2012. Example-based synthesis of 3D object arrangements. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–11.

Matthew Fisher, Manolis Savva, Yangyan Li, Pat Hanrahan, and Matthias Nießner. 2015. Activity-centric scene synthesis for functional 3D scene modeling. *ACM Transactions on Graphics* 34, 6 (2015), 1–13.

Qiang Fu, Xiaowu Chen, Xiaotian Wang, Sijia Wen, Bin Zhou, and Hongbo Fu. 2017. Adaptive synthesis of indoor scenes via activity-associated object relation graphs. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–13.

Ghost Town Games. 2016. Overcooked. https://store.steampowered.com/app/448510/Overcooked/.

Anindita Ghosh, Noshaba Cheema, Cennet Oguz, Christian Theobalt, and Philipp Slusallek. 2021. Synthesis of compositional animations from textual descriptions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1396–1406.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*. PMLR, 1263–1272.

Chuan Guo, Xinxin Zuo, Sen Wang, Shihao Zou, Qingyao Sun, Annan Deng, Minglun Gong, and Li Cheng. 2020. Action2motion: Conditioned generation of 3d human motions. In *Proceedings of the 28th ACM International Conference on Multimedia*. 2021–2029.

Xiaojie Guo and Liang Zhao. 2022. A systematic survey on deep generative models for graph generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

Mohamed Hassan, Duygu Ceylan, Ruben Villegas, Jun Saito, Jimei Yang, Yi Zhou, and Michael J Black. 2021a. Stochastic scene-aware motion prediction. In *ICCV*. 11374–11384.

Mohamed Hassan, Partha Ghosh, Joachim Tesch, Dimitrios Tzionas, and Michael J Black. 2021b. Populating 3D Scenes by Learning Human-Scene Interaction. In *CVPR*. 14708–14718.

W Keith Hastings. 1970. Monte Carlo sampling methods using Markov chains and their applications. (1970).

Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. 2020. Graph2plan: Learning floorplan generation from layout graphs. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 118–1.

Haroon Idrees, Amir R Zamir, Yu-Gang Jiang, Alex Gorban, Ivan Laptev, Rahul Sukthankar, and Mubarak Shah. 2017. The THUMOS challenge on action recognition for videos "in the wild". *Computer Vision and Image Understanding* 155 (2017), 1–23.

Ananya Ipsita, Levi Erickson, Yangzi Dong, Joey Huang, Alexa K Bushinski, Sraven Saradhi, Ana M Villanueva, Kylie A Peppler, Thomas S Redick, and Karthik Ramani. 2022. Towards Modeling of Virtual Reality Welding Simulators to Promote Accessible and Scalable Training. In *CHI Conference on Human Factors in Computing Systems*. 1–21.

Jingwei Ji, Ranjay Krishna, Li Fei-Fei, and Juan Carlos Niebles. 2020. Action genome: Actions as compositions of spatio-temporal scene graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10236–10247.

Baoxiong Jia, Yixin Chen, Siyuan Huang, Yixin Zhu, and Song-chun Zhu. 2020. Lemma: A multi-view dataset for learning multi-agent multi-task activities. In *European Conference on Computer Vision*. Springer, 767–786.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2018. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*. PMLR, 2323–2332.

Hanseob Kim, Ghazanfar Ali, and Jae-In Hwang. 2021. ASAP: Auto-generating Storyboard And Previz with Virtual Humans. In *2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE, 316–320.

Vladimir G Kim, Siddhartha Chaudhuri, Leonidas Guibas, and Thomas Funkhouser. 2014. Shape2pose: Human-centric shape analysis. *ACM Transactions on Graphics* 33, 4 (2014), 1–12.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680.

Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. 2006. Motion patches: building blocks for virtual environments annotated with motion data. In *ACM SIGGRAPH 2006 Papers*. 898–906.

Seunghwan Lee, Phil Sik Chang, and Jehee Lee. 2022. Deep Compliant Control. In *ACM SIGGRAPH 2022 Conference Proceedings*. Article 23, 9 pages.

Ang Li, Meghana Thotakuri, David A Ross, João Carreira, Alexander Vostrikov, and Andrew Zisserman. 2020. The ava-kinetics localized human actions video dataset. *arXiv preprint arXiv:2005.00214* (2020).

Changyang Li, Wanwan Li, Haikun Huang, and Lap-Fai Yu. 2022. Interactive augmented reality storytelling guided by scene semantics. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–15.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324* (2018).

Zelun Luo, Zane Durante, Linden Li, Wanze Xie, Ruochen Liu, Emily Jin, Zhuoyi Huang, Lun Yu Li, Jiajun Wu, Juan Carlos Niebles, et al. 2022. MOMA-LRG: Language-Refined Graphs for Multi-Object Multi-Actor Activity Parsing. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Zelun Luo, Wanze Xie, Siddharth Kapoor, Yiyun Liang, Michael Cooper, Juan Carlos Niebles, Ehsan Adeli, and Fei-Fei Li. 2021. MOMA: Multi-Object Multi-Actor Activity Parsing. *Advances in Neural Information Processing Systems* 34 (2021), 17939–17955.

Rui Ma, Honghua Li, Changqing Zou, Zicheng Liao, Xin Tong, and Hao Zhang. 2016. Action-driven 3D indoor scene evolution. *ACM Trans. Graph.* 35, 6 (2016), 173–1.

Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. 2011. Interactive furniture layout using interior design guidelines. *ACM transactions on graphics (TOG)* 30, 4 (2011), 1–10.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. 1953. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics* 21, 6 (1953), 1087–1092.

Nawmal. 2019. Nawmal. https://www.nawmal.com/

Mathis Petrovich, Michael J Black, and Gül Varol. 2021. Action-conditioned 3d human motion synthesis with transformer vae. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10985–10995.

Sören Pirk, Vojtech Krs, Kaimo Hu, Suren Deepak Rajasekaran, Hao Kang, Yusuke Yoshiyasu, Bedrich Benes, and Leonidas J Guibas. 2017. Understanding and exploiting object interaction landscapes. *ACM Transactions on Graphics (TOG)* 36, 3 (2017), 1–14.

Manolis Savva, Angel X Chang, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. 2014. SceneGrok: Inferring action maps in 3D environments. *ACM Transactions on Graphics* 33, 6 (2014), 1–10.

Manolis Savva, Angel X Chang, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. 2016. Pigraphs: learning interaction snapshots from observations. *ACM Transactions on Graphics* 35, 4 (2016), 1–12.

Dian Shao, Yue Zhao, Bo Dai, and Dahua Lin. 2020. Finegym: A hierarchical video dataset for fine-grained action understanding. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2616–2625.

Martin Simonovsky and Nikos Komodakis. 2018. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*. Springer, 412–422.

Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. 2019. Neural state machine for character-scene interactions. *ACM Transactions on Graphics* 38, 6 (2019), 209–1.

Jiahui Sun, Wenming Wu, Ligang Liu, Wenjie Min, Gaofeng Zhang, and Liping Zheng. 2022. WallPlan: synthesizing floorplans by learning to generate wall graphs. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–14.

Tomu Tahara, Takashi Seno, Gaku Narita, and Tomoya Ishikawa. 2020. Retargetable AR: Context-aware augmented reality in indoor scenes based on 3D scene graph. In *2020 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE, 249–255.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

He Wang, Sören Pirk, Ersin Yumer, Vladimir G Kim, Ozan Sener, Srinath Sridhar, and Leonidas J Guibas. 2019b. Learning a Generative Model for Multi-Step Human-Object Interactions from Videos. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 367–378.

Jiashun Wang, Huazhe Xu, Jingwei Xu, Sifei Liu, and Xiaolong Wang. 2021a. Synthesizing long-term 3d human motion and interaction in 3d scenes. In *CVPR*. 9401–9411.

Jingbo Wang, Sijie Yan, Bo Dai, and Dahua Lin. 2021b. Scene-aware generative network for human motion synthesis. In *CVPR*. 12206–12215.

Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X Chang, and Daniel Ritchie. 2019a. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–15.

Kai Wang, Manolis Savva, Angel X Chang, and Daniel Ritchie. 2018. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Zan Wang, Yixin Chen, Tengyu Liu, Yixin Zhu, Wei Liang, and Siyuan Huang. 2022. Humanise: Language-conditioned human motion generation in 3d scenes. *arXiv preprint arXiv:2210.09729* (2022).

Sifan Ye, Yixing Wang, Jiaman Li, Dennis Park, C. Karen Liu, Huazhe Xu, and Jiajun Wu. 2022. Scene Synthesis from Human Motion. In *SIGGRAPH Asia 2022 Conference Papers*. Article 26, 9 pages.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*. PMLR, 5708–5717.

Lap-Fai Yu, Sai Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley Osher. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.* 30, 4 (2011), 86.

Jia-Qi Zhang, Xiang Xu, Zhi-Meng Shen, Ze-Huan Huang, Yang Zhao, Yan-Pei Cao, Pengfei Wan, and Miao Wang. 2021. Write-An-Animation: High-level Text-based Animation Editing with Character-Scene Interaction. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 217–228.

Siwei Zhang, Yan Zhang, Qianli Ma, Michael J Black, and Siyu Tang. 2020c. PLACE: Proximity learning of articulation and contact in 3D environments. In *2020 International Conference on 3D Vision (3DV)*. IEEE, 642–651.

Yan Zhang, Mohamed Hassan, Heiko Neumann, Michael J Black, and Siyu Tang. 2020a. Generating 3d people in scenes without people. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 6194–6204.

Zaiwei Zhang, Zhenpei Yang, Chongyang Ma, Linjie Luo, Alexander Huth, Etienne Vouga, and Qixing Huang. 2020b. Deep generative modeling for scene synthesis via hybrid representations. *ACM Transactions on Graphics (TOG)* 39, 2 (2020), 1–21.