Hierarchical Motion Understanding via Motion Programs

Sumith Kulal* Stanford University
Jiayuan Mao* MIT
Alex Aiken Stanford University
Jiajun Wu Stanford University

Abstract

Current approaches to video analysis of human motion focus on raw pixels or keypoints as the basic units of reasoning. We posit that adding higher-level motion primitives, which can capture natural coarser units of motion such as "backswing" or "follow-through", can be used to improve downstream analysis tasks. This higher level of abstraction can also capture key features, such as loops of repeated primitives, that are currently inaccessible at lower levels of representation. We therefore introduce Motion Programs*, a neuro-symbolic, program-like representation that expresses motions as a composition of high-level primitives. We also present a system for automatically inducing motion programs from videos of human motion and for leveraging motion programs in video synthesis. Experiments show that motion programs can accurately describe a diverse set of human motions and the inferred programs contain semantically meaningful motion primitives, such as arm swings and jumping jacks. Our representation also benefits downstream tasks such as video interpolation and video prediction and outperforms off-the-shelf models. We further demonstrate how these programs can detect diverse kinds of repetitive motion and facilitate interactive video editing.*

1. Introduction

Most current video analysis architectures operate on either pixel-level or keypoint-level transitions that are good at local and short-term motions of human bodies but do not handle higher-level spatial and longer-lasting temporal structures well. We posit that incorporating higher-level reasoning of motion primitives and their relationships enables better video understanding and improved downstream task performance: after seeing a person perform jumping jacks a few times, it's reasonable to predict that they may repeat the same action for a few more iterations.

We thus introduce a hierarchical motion understanding framework with three levels of abstraction, as shown in Figure 1:

- Level 1 (Figure 1b) is **Keypoints**, which are the basis of most current computer vision systems. This level provides low-level priors such as temporal smoothing.
- Level 2 (Figure 1c) is **Concrete Motion Programs**, a programmatic representation of motion consisting of a sequence of atomic actions or *motion primitives*, each describing a set of frames for a single action. This level provides priors on primitive level motion.
- Level 3 (Figure 1d) is **Abstract Motion Programs**, which describe higher-level groupings of the concrete primitives, such as patterns of repetitive motion. This level provides priors on sequences of primitives and their patterns.

We describe the concrete and abstract levels with simple domain specific languages (DSLs). The Concrete DSL consists of motion primitives such as stationary, linear, and circular movement with necessary spatial and temporal motion parameters. The Abstract DSL includes for-loops for capturing higher-level groupings of primitives. Along with the DSLs, we propose an algorithm to automatically synthesize motion programs from videos. For the sake of brevity, we often refer to motion programs as simply programs.

To demonstrate the advantages of motion programs, we apply our methods for synthesizing motion programs from videos to three applications: video interpolation (Section 4.2), repetitive segment detection (Section 4.3) and video prediction (Section 4.4). For video interpolation, we demonstrate that when combined with state-of-the-art image synthesis models, such as SepConv [23], motion programs can achieve higher quality interpolations with bigger gaps between frames than previous approaches. For repetitive segment detection, we demonstrate that motion programs can detect and extract repetitive segments of human motion. For video prediction, we show that motion programs can perform longer-range video prediction than HierchVid [29] and HistryRep [32] by identifying repetitive motions.

To summarize, our contributions are:

- We introduce a hierarchical motion understanding framework using motion programs.
- We present an algorithm for automatically synthesizing motion programs from raw videos.

*Equal contribution. Email: sumith@cs.stanford.edu.
Webpage: https://sumith1896.github.io/motion2prog.

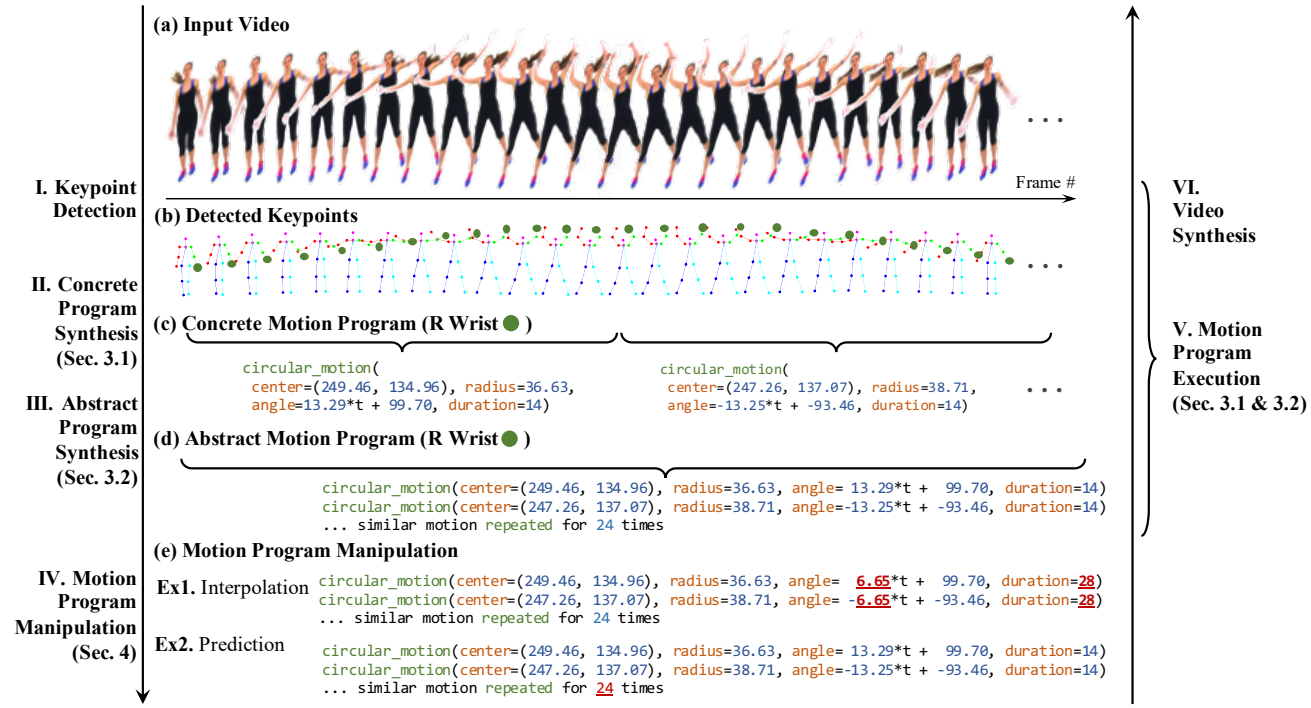

Figure 1. Overview of our hierarchical motion understanding framework. We start with low-level keypoints extracted from raw videos and build higher level motion understanding. Higher levels of abstraction consist of a sequence of motion primitives and their repetitive patterns. Similar motion in repetitive patterns are captured using a probabilistic for-loop body (refer Figure 3).

- By incorporating motion programs into video synthesis pipelines, we demonstrate higher performance in video interpolation, video prediction and also show how to detect and extract repetitive motion segments.

2. Related Work

We briefly summarize the most relevant related work in capturing and understanding images and video.

Program-like representations for visual data. The idea of using programs to represent visual data dates back to the early computer graphics literature on using procedural models to model biological structures [17], 3D shapes [14, 26, 27], and indoor scenes [30, 15, 24]. Recent works integrate the representational power of deep networks with program-like structures in visual data modeling, which enables the application of high-level program-like structures in more complex and noisy environments such as hand-drawn sketches [8] and even natural images [33, 21]. These works have all focused on extracting programmatic descriptions of images, whereas our main contribution is in developing and applying programmatic descriptions of motion to videos.

Our model also relates to works on using parameterized curves [25, 10, 11] or motion segments that can be recombined [13, 1] for modeling and editing motions. Compared with these works, we focus on inferring program-like representation of human motion from raw videos, and introduce abstract motion programs to describe higher-level groupings of low-level motions, which enables us to capture patterns of repetitive motion. As a result, the applications of such representations are different, with our programmatic representation being useful for multiple video manipulation tasks.

Video synthesis. Substantial progress has recently been made on video processing tasks including video interpolation [19, 12, 18, 23], video extrapolation [29, 7], and video retiming [20, 6]. These methods use deep convolutional neural networks (CNNs) to extract spatial-temporal features of videos. These features are sent to task-specific networks to make predictions of intermediate frames (in video interpolation) or future frames (in video extrapolation). Various methods have been proposed to improve the results by predicting flows [18], predicting blending kernels [23], making future predictions in keypoint spaces [29], and using variational inference to model video stochasticity [7]. Our approach works at a higher level of abstraction and captures structure on longer time scales. We show that the two classes of work can be combined to yield improved results for video interpolation.

3. Hierarchical Motion Understanding

Our hierarchical motion understanding framework has three levels of increasing abstraction. The first level is repre-

senting motion as a sequence of keypoints. For the next two levels, we use programmatic representations of motion via two simple domain-specific languages (DSLs).

In Section 3.1, we describe the synthesis and execution of *concrete motion programs* (Level 2). In Section 3.2, we describe synthesis and execution of *abstract motion programs* (Level 3).

Level 1—Keypoints. Representing human motion as a sequence of keypoints is a well-studied problem. Several high-quality, off-the-shelf tools such as AlphaPose [9] and OpenPose [4] take raw video as input and output a sequence of 2D keypoints for each body joint. Sequences of keypoints for a single joint are our first level of abstraction.

Level 2—Concrete Motion Programs. A concrete motion program is a sequence of motion primitives, where each primitive is either circular, linear, or stationary (not moving) motion of a specific keypoint for a fixed time duration. Table 1 gives a grammar for the DSL. These programs are *concrete* because the specific kind of motion (e.g., linear) of each primitive in the sequence is explicit as are all the parameters needed to fully describe the motion.

More specifically, we define three parameterized motion primitives: a circular primitive (`circle`), a linear primitive (`line`) and a stationary primitive (`stationary`). Each primitive has spatial parameters, such as `center` and `radius` for circular motion, temporal motion parameters such as `velocity` and `start_angle`, and a time duration. In practice, we have found these three simple motion primitives are sufficient to cover a wide variety of use cases. The framework could easily extended with more primitives or more expressive primitives (such as spline curves). The execution model of concrete programs is the obvious one: The keypoint traces out each primitive p in order, beginning in the starting configuration for p and for p's specified amount of time.

Level 3—Abstract Motion Programs. Abstract motion programs (see Table 2) consist of sequences and loops of primitive distributions over start, middle, and end points of the motion. Note that the start-middle-end representation does not commit to the type of motion (e.g., circular *vs*. linear) and in fact a specific distribution might include instances of multiple kinds of motion. To execute an abstract primitive p, we sample the start, middle and end points (as well as other parameters, such as duration) of p's distribution, pick the concrete primitive c that best fits the sample, and then execute c. Sequences and loops of abstract primitives are then executed by repeating this process for each abstract primitive in the execution sequence. The start-middle-end representation could be extended with more points, but at least three points are needed to distinguish linear from circular motion.

Abstract motion programs concisely capture complex categories of human motion. For example, "jumping jacks" are easily expressed by a loop with a body consisting of a sequence of primitives describing the up and down motion of a single jumping jack. The fact that the primitives are distributions captures the natural variation that arises in repeated jumping jacks—no two will be exactly the same, but all will be similar within some bounds.

Table 1. DSL of Concrete Programs (Level 2) for describing motion as a sequence of motion primitives.

Program	$\longrightarrow$	Prim; . . . ; Prim
Prim	$\longrightarrow$	`circle` (center: Point, radius: R, angle=(vel: R, start: R), time: Int)
Prim	$\longrightarrow$	`line` (x=(vel: R, start: R), y=(slope: R, intercept: R), time: Int)
Prim	$\longrightarrow$	`stationary`(point: Point, time: Int)
Point	$\longrightarrow$	(x: R, y: R)
Int	$\longrightarrow$	a positive integer
R	$\longrightarrow$	a real number

Table 2. DSL of Abstract Programs (Level 3) for describing motion as a sequence of probabilistic motion primitives with loops.

Program	$\longrightarrow$	Stmt; . . . ; Stmt
Stmt	$\longrightarrow$	DetPrim | For-Loop (iter: Int) { ProbPrim; . . . ; ProbPrim }
DetPrim	$\longrightarrow$	{<start, middle, end>: <Point, Point, Point>, time: Int}
ProbPrim	$\longrightarrow$	{<start, middle, end>: Gaussian(<Point, Point, Point>),), time: FreqDist(Int)}
Point	$\longrightarrow$	(x: R, y: R)
Int	$\longrightarrow$	a positive integer
R	$\longrightarrow$	a real number

3.1. Concrete Motion Programs

We now describe how we connect the different levels of our hierarchy. Given a sequence of keypoints we synthesize a concrete motion program, and given a concrete motion program we identify repeating patterns in the sequence of concrete primitives to synthesize the distributions and loops of an abstract motion program. We begin with concrete motion programs.

Synthesizing single primitives. To fit a single circular primitive to a sequence of keypoints, we first find the best-fit circle of the sequence [3]. We then project the keypoints onto this best-fit curve and find temporal motion parameters (`velocity`, `start_angle`) that best approximate the motion of the projected points. The problem can also be solved by jointly optimizing for all parameters, but in practice we observed that this two step approach produced equivalent results with significantly faster running times. We run this process for linear and stationary motion and pick the concrete primitive that minimizes the $\mathcal{L}_2$ error. Some examples of single primitive synthesis are given in Figure 2.

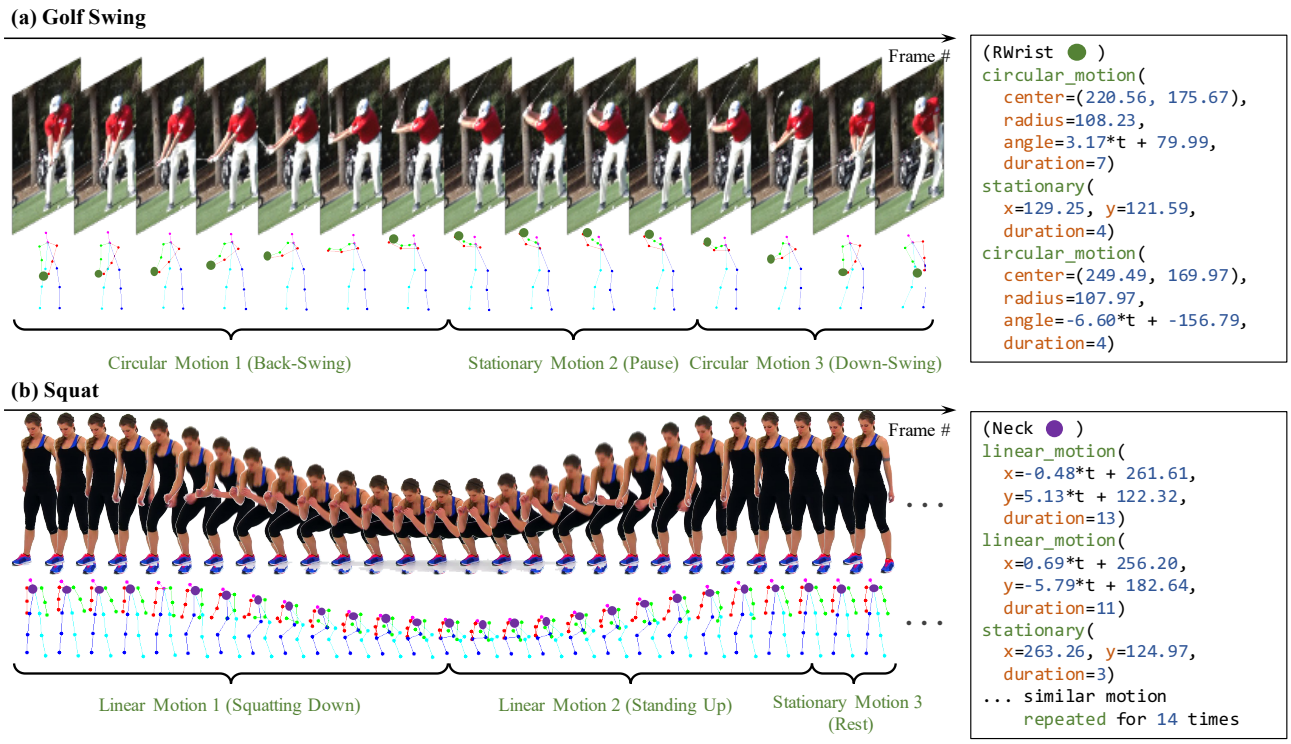

Figure 2. Examples of synthesized motion primitives. a) This golf swing has three primitives: back-swing, pause and down-swing. b) The squats sequence has a repeating subsequence of three similar primitives: squating down, standing up and a brief rest in the standing pose.

Segmentation and synthesis. A long video is rarely well-described by a single concrete primitive. To synthesize a sequence of primitives, we must segment the keypoint sequence and synthesize a primitive for each segment. We use dynamic programming to segment the input sequence at the best possible locations (i.e., the ones that minimize overall error) and fit individual primitives to each segment, as illustrated in Figure 2. The recurrence relation for the best fit sequence of primitives for the first n keypoints is given as

$$\text{Error}_\text{n} = \min_{k<n} \left[\text{Error}_\text{k} + \text{fit}(\text{keypoints}[k:n]) + \lambda\right]. \quad (1)$$

Here, Error_k is the error incurred by the best fit sequence of primitives for the first k keypoints and $\text{fit}(\text{keypoints}[k:n])$ returns the error incurred by the best-fit single primitive for keypoints from k to n. The parameter λ is a regularization term that controls the granularity of fitted primitives. Note that $\lambda = 0$ results in a degenerate fit with every point labeled as an independent primitive and $\lambda \to \infty$ results in a single primitive. Hence, in practice, we adaptively choose λ at runtime to match the range of the keypoint motion by setting λ proportional to the covariance of keypoints in a fixed window centered on the current keypoint.

Multiple keypoints. The segmentation and synthesis algorithm can be generalized from one keypoint trajectory to multiple keypoint trajectories by jointly finding the best single segmentation across all the trajectories by minimizing the sum of errors across all keypoints. This approach enforces the useful invariant that programs for all keypoints transition to new primitives simultaneously. Note that this is the only place where we have found it useful to jointly consider all keypoint trajectories; otherwise motion programs are constructed per keypoint.

Putting it all together, some examples of synthesized primitives after segmentation generated from golf swings and squats are shown in Figure 2.

3.2. Abstract Motion Programs

The translation from a concrete motion program to an abstract motion program is done in three steps: converting concrete primitives to abstract primitives, loop detection, and loop synthesis.

Concrete to abstract translation. We first replace each concrete primitive with a deterministic abstract primitive. We do this by obtaining the start, middle and end points of the concrete primitive along with the duration of execution.

Loop detection. In the second step we detect loops in the sequence of abstract primitives. We use a sliding window of W primitives and attempt to fit loops with different numbers of abstract primitives l in the loop body for each window. For each candidate loop body size l and $W = \langle w_0, \ldots, w_n \rangle$ we first group together all the deterministic primitives that

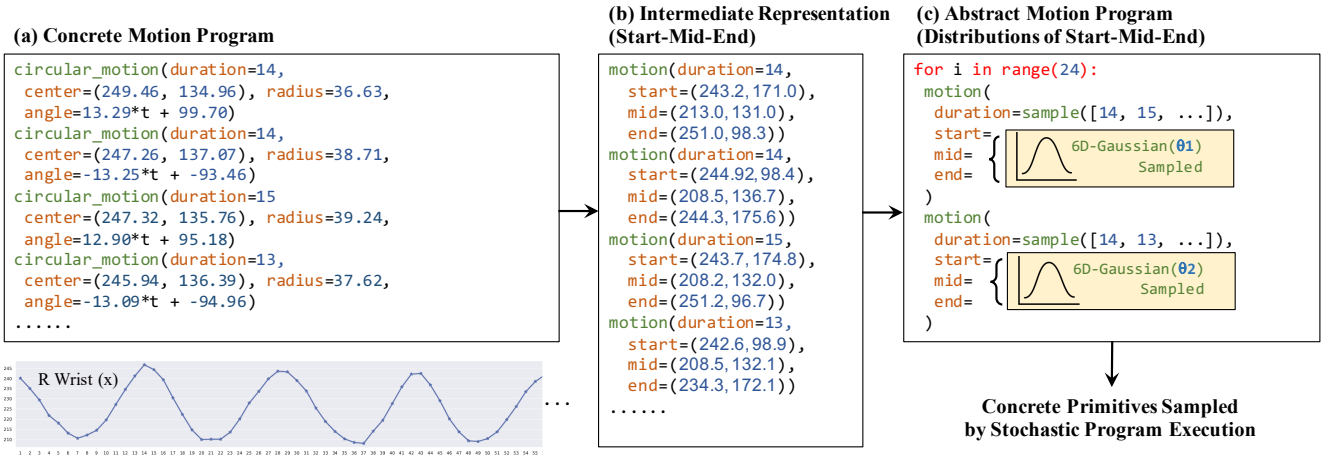

Figure 3. Illustration of rolling up 6 repetitive (alternating) statements into a for-loop of body size 2. We first translate concrete primitives to deterministic abstract primitives and then synthesize for-loops with probabilistic primitives in the body. Concrete primitives are sampled from the probabilistic abstract primitives during execution.

would be in the ith abstract primitive of the loop body:

$$S_i = \{w_j | j\%l = i\} \text{ for } 0 \leq i < l. \quad (2)$$

Now $S_0, \ldots, S_{l-1}$ is easily converted to a loop by fitting a Gaussian distribution to each S_i to create an abstract primitive of the loop body. Note that we could not directly learn such a distribution of parameters for concrete primitives since each set could have more than one primitive type.

We define the quality of fit for a loop as the average of the Gaussian covariance norms across S_i. We threshold the quality and below a certain value mark it as a successful loop detection. To find the maximal loop, we dynamically increase the window size to include more primitives till covariance norms are still below threshold. We search over increasing loop body lengths l until either a loop is successfully detected or we exceed a maximum loop body size.

Loop synthesis. As the last step, we replace intervals of abstract primitives with the detected for-loop. The number of iterations is set according to the number of lines rolled up and the loop body size. An illustration of translating concrete primitives to abstract primitives and subsequent loop detection is presented in Figure 3.

4. Experiments

To evaluate the expressive power of both concrete and abstract motion programs, we first evaluate the accuracy of synthesized motion programs (Section 4.1). Second, we evaluate the motion programs on video interpolation and compare it with state-of-the-art models (Section 4.2). Third, we present results for detecting and extracting repetitive motion from long exercise videos (Section 4.3). Finally, we demonstrate interactive manipulation of repetitive motion in videos using abstract motion programs and compare it to video prediction baselines (Section 4.4).

4.1. Motion Program Induction

Data. We use the GolfDB dataset [22] for evaluating the representative power of concrete programs. We extract videos from swing start to swing end from all 242 face-on real-time golf swing videos. We use a 50-50 train-test split and scale each frame to 512×512.

Analysis. We synthesize concrete motion programs for the train set of GolfDB and compare the synthesized programs with ground-truth poses. On average, we synthesize 3.33 primitives per video where the length of videos averages around 58 frames. In the majority of these videos, the primitives semantically correspond to back-swing, down-swing and follow-through with some stationary primitives at the beginning and the end. We analyse the efficiency, accuracy and temporal consistency of program representation. For efficiency, we compare the average number of parameters needed to represent the programs and the input pose sequence. Programs needed 253 parameters per video, which is much lower compared to 1,623 parameters needed by input poses. For accuracy, we compute the keypoint difference metric (KD) of the program-encoded pose sequence with the input pose; KD is the average distance of keypoints with the ground-truth keypoints in pixel space. We observe that the program-encoded poses differ from the ground-truth poses by 6.25 pixels, an error of only 2.44%. This indicates that motion programs are good approximates of input pose sequence. Due to inherent noise in pose detection, the input poses could at times be noisy. We observe that the motion program representation smoothens out this noise to some degree by. To validate this, we compute the maximum difference in pixels of the pose of a joint in two adjacent frames. We observe that the input pose sequence has a difference of 100.10 pixels while program-encoded pose sequence are much smoother (76.15 pixels).

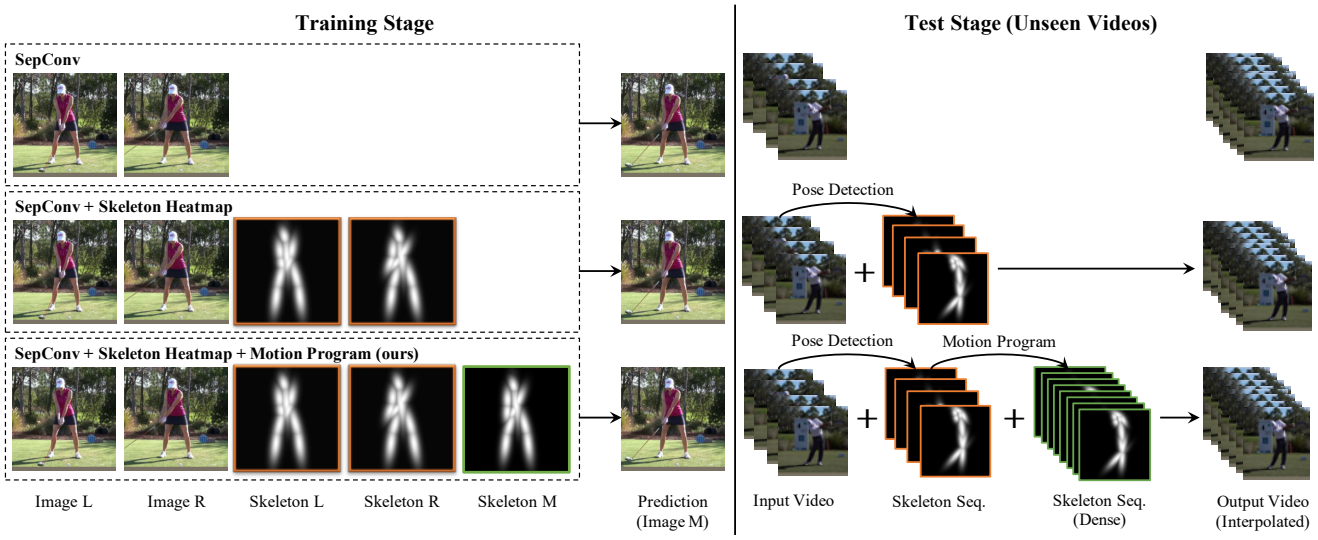

Figure 4. Evaluation of three video interpolation variants: Top row (SepConv [23]), middle row (SepConv w/ heatmaps) and bottom row (SepConv w/ programs) which is our model where poses for intermediate frames are supplied by motion programs. Middle row (SepConv w/ heatmaps) is evaluated as an ablation study.

4.2. Video Interpolation

We leverage motion programs to perform video interpolation and demonstrate the benefit of having primitive-level information in such tasks.

Data. We evaluate on two datasets, GolfDB (as described above) and PoseWarp [2]. For the PoseWarp dataset, we stitch together continuous frames to obtain 105 videos. We scale each video to 512x512 and use the training splits specified in the original paper.

Setup. We take the standard SepConv [23] architecture and modify it to incorporate the program information as shown in Figure 4. We train three variants of SepConv: i) SepConv, as defined in the original paper, that predicts the interpolated frame given left and right frames; ii) SepConv with input pose information of both frames (SepConv w/ heatmaps); iii) SepConv with both input and target pose information (SepConv w/ programs). For the last model, at test time, the the target pose information is supplied by a motion program. We synthesize a motion program on input video and generate intermediate poses from executing the program with a finer granularity of time. Note that this information is otherwise not available. We are interested in the first and third model, where the second model is also studied as an ablation.

For evaluation, we generate programs on the test set of both datasets. For each primitive generated, we sub-sample the frames at different rates (2x/4x/8x) and try to reconstruct the original video by interpolating the remaining frames. Performing 4x interpolation is more challenging than 2x and 8x is even more challenging.

Metrics. We use the standard PSNR, SSIM [31], and LPIPS [34] image metrics to estimate the fidelity of synthesized videos. We also compute the keypoint difference (KD) metric as before, but now we find the average distance of keypoints detected in synthesized video with the keypoints detected in the ground-truth video. We present KD results for the major motion joints: left elbow, left wrist, right elbow, and right wrist.

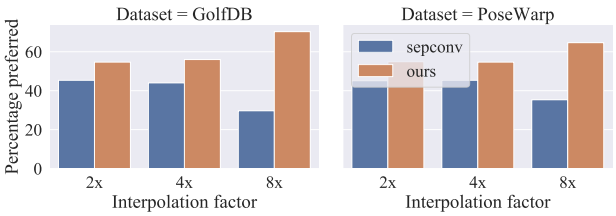

Figure 5. Human evaluation of video interpolation across different rates on both datasets. Incorporating motion programs significantly improves upon the baseline for 8x interpolation.

For evaluating the quality of synthesized videos, we also perform human preference studies on Amazon Mechanical Turk (AMT). We randomly showed videos synthesized from both the baseline and our method and ask humans 'Which video looks more realistic?' For each task, 30 subjects were each asked to vote on 30 random videos from the test set.

Results. The quantitative results are shown in Table 3. We observe that the performance of our model (SepConv w/ programs) is comparable to SepConv at 2x interpolation and improves significantly for 8x interpolation on both datasets. Our model retains higher-level structure better as indicated by the KD metrics. It also generates realistic videos as validated by the human evaluation shown in Figure 5. Figure 6 highlights the qualitative improvements, our model can generate more realistic hands for sparse 8x interpolation.

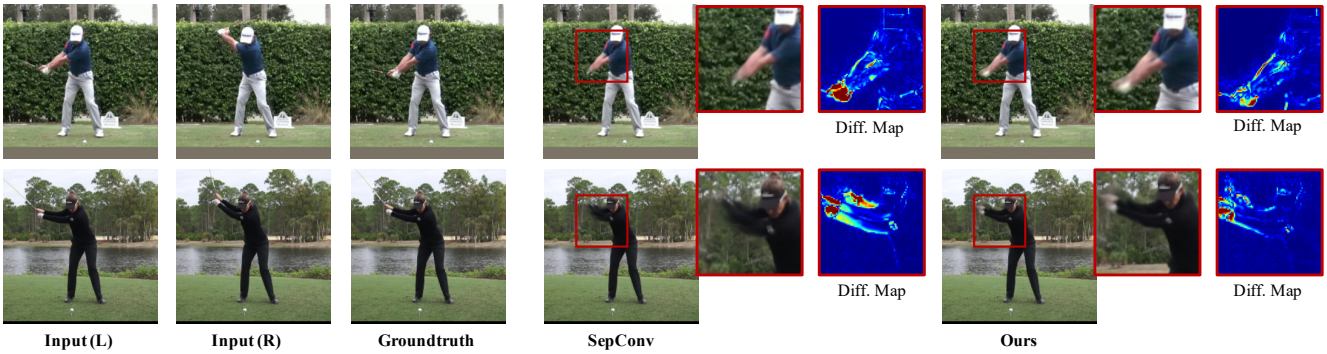

Figure 6. Qualitative results for 8x video interpolation. We observe that using program information results in uniformly higher image quality.

Table 3. Comparison of metrics for video interpolation of 2x / 4x / 8x subsampling on the GolfDB and PoseWarp test sets. Metrics include keypoint difference in left/right elbow and wrist (lower is better), PSNR, SSIM (higher is better), and LPIPS (lower is better).

Method	Dataset	Keypoint Difference↓					Perceptual Metrics		
		l-elbow	l-wrist	r-elbow	r-wrist	average	PSNR↑	SSIM↑	LPIPS↓
SepConv [23]	PoseWarp 2x	2.556	3.606	2.330	3.124	1.791	32.285	0.967	0.023
SepConv w/ heatmaps	PoseWarp 2x	2.357	3.490	2.267	2.910	1.788	32.246	**0.968**	0.025
SepConv w/ programs	PoseWarp 2x	**2.238**	**2.982**	**1.965**	**2.552**	**1.650**	**32.699**	0.964	**0.021**
SepConv [23]	GolfDB 2x	3.263	3.348	**3.077**	3.048	**2.308**	34.860	0.981	0.017
SepConv w/ heatmaps	GolfDB 2x	3.395	3.289	3.211	2.935	2.355	35.046	0.982	0.017
SepConv w/ programs	GolfDB 2x	**3.210**	**3.263**	3.325	**2.900**	2.432	**35.597**	**0.982**	**0.015**
SepConv [23]	PoseWarp 4x	3.319	4.625	2.926	3.933	2.151	**30.834**	**0.955**	0.034
SepConv w/ heatmaps	PoseWarp 4x	3.011	4.305	2.708	3.699	2.088	30.82	**0.955**	0.036
SepConv w/ programs	PoseWarp 4x	**2.788**	**3.441**	**2.495**	**3.068**	**1.938**	30.344	0.951	**0.032**
SepConv [23]	GolfDB 4x	4.987	5.791	4.620	4.797	3.598	30.488	0.958	0.035
SepConv w/ heatmaps	GolfDB 4x	4.927	5.351	4.670	4.783	3.532	30.606	0.959	0.034
SepConv w/ programs	GolfDB 4x	**4.466**	**4.704**	**4.085**	**4.093**	**3.091**	**30.990**	**0.959**	**0.031**
SepConv [23]	PoseWarp 8x	5.520	9.349	4.132	7.527	3.275	**27.659**	**0.923**	0.059
SepConv w/ heatmaps	PoseWarp 8x	5.367	9.214	4.672	8.425	3.411	27.666	0.923	0.06
SepConv w/ programs	PoseWarp 8x	**4.371**	**6.611**	**3.955**	**6.475**	**2.840**	27.587	0.92	**0.054**
SepConv [23]	GolfDB 8x	10.762	15.852	8.340	14.075	6.900	25.672	0.908	0.071
SepConv w/ heatmaps	GolfDB 8x	10.914	16.135	8.839	15.484	7.317	25.918	0.913	0.068
SepConv w/ programs	GolfDB 8x	**8.496**	**11.134**	**7.134**	**9.929**	**5.669**	**26.563**	**0.914**	**0.062**

4.3. Extracting Repetitive Segments

We now evaluate motion programs' ability on automatically extracting repetitive segments from long action clips.

Data. We use three long (15+ minutes) exercise videos from YouTube, two with cardio and one with Taichi routines. We manually annotated the videos with ground-truth labels for repetitive action segments.

Results. We generate abstract motion programs on full videos and evaluated the precision and recall of the detected for-loops. We manually annotated the videos with ground-truth labels to compute these metrics. We mark a loop as successfully extracted if the IoM (intersection over minimum) of the detected interval and ground-truth interval is greater than 0.5. The results are in Table 4. We present samples of repetitive segments extracted in the project webpage. We also present additional qualitative examples for repetitive structure extracted from the AIST++ dance database [16, 28].

Error analysis. We observed a few failure cases that contribute to the less than perfect precision and recall. First, our synthesis algorithm is not good at capturing fine small motions. The problem originates with pose detectors, which have low signal-to-noise ratio for small motions. For example, cardio00 has several small jogging motions which we do not mark as loops leading to lower recall values. Synthesis also sometimes produces poor results due to bad keypoint detection from body occlusions. Finally, one would expect that complex 3D human actions in the z-plane are not well-described by 2D keypoints. We present a more detailed analysis with examples in the project webpage.

Table 4. Comparing the number of detected segments (# det) with ground-truth segments (# gt) on three long exercise videos.

Video ID	Duration	# det	# gt	Precision	Recall
cardio00	37:10	36	62	88.57	58.06
cardio01	17:51	19	21	69.35	90.47
taichi02	23:22	23	34	100	67.64

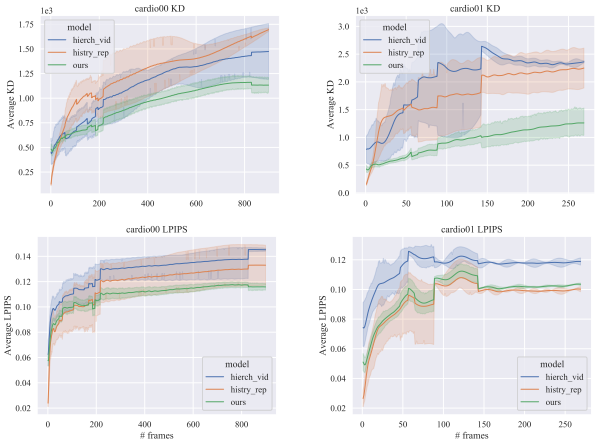

Figure 7. Comparison of HierchVid [29] (blue), HistryRep [32] (orange) and ours (green) on video prediction. Top row: KD↓ (keypoint difference), bottom row: LPIPS↓ on loops extracted from cardio00 and cardio01.

4.4. Video Prediction

Once a set of for-loops has been detected in a video, motion programs enable indefinite extrapolation by executing the synthesized for-loops for additional iterations. This enables an interactive video editing platform, where one could expand or contract the repetitive section. In this section, we evaluate how well the for-loop based video prediction compare to standard baselines.

Data. For each long video used in the previous section, we extract a dataset of all the detected for-loops. We use an 80-20 train-test split.

Setup. For each dataset, we use the training set to train our baselines, HierchVid [29] and HistryRep [32]. HierchVid is an LSTM-based future pose predictor while HistryRep uses an attention-based model. At test time, we use the first half of the videos as input and compare the future poses generated with the second half as ground-truth. We also use first half as input to generate a motion program with for-loop and unroll it indefinitely to generate future poses. We then synthesize future frames from poses for both the methods using the Everybody Dance Now [5] GAN and compare it with the ground-truth frames.

Metrics. Similar to video interpolation, we use the PSNR, SSIM and LPIPS image metrics when ground truth frames are available. We also compute the keypoint difference (KD) with the ground-truth poses. We average our results over 10 runs. Since predicting future frames is a stochastic task

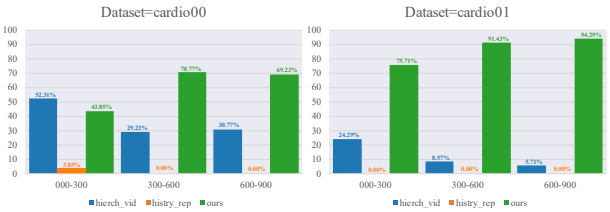

Figure 8. Human evaluation of video prediction across different future frame slices on two datasets. We observe that extrapolating via motion programs leads to more consistent future prediction of repetitive segments

and direct comparison to ground-truth might be an incomplete analysis, we also perform human preference studies on AMT. We randomly showed videos synthesized from both the methods and ask humans 'Which video looks more similar to the ground-truth?'. For each dataset, 10 subjects were each asked to vote for all the videos from the test set. To have a granular understanding, we did three studies, comparing the first 900 frames in sets of 300 frames.

Results. We present quantitative results in Figure 7 where we plot the average KD and LPIPS metrics on the two datasets against the number of future frames predicted. We observe that as the number of future frames increases, the gap between the baselines and our model increases. This is due to the fact that both HierchVid and HistryRep perform sequence modelling and do not have higher level knowledge of repetitions. After the first few frames, the error compounds and the model diverges to performing a different action. This fact is reinforced by the human evaluation study, results of which are in Figure 8. On the cardio00 dataset, HierchVid does well on the first few frames, though in the later frames our model is more consistent with the ground-truth. We present additional results on more metrics on our webpage.

5. Discussion

We have presented a hierarchical motion understanding framework based on motion programs. From the input video, we can automatically build primitive-level understanding while also capturing higher-level repetitive structures. Experiments show that these programs can capture a wide range of semantically meaningful actions, enhance human motion understanding, and enable applications such as video synthesis for computer animation. Like all other visual content generation methods, we urge users of our models to be aware of potential ethical and societal concerns and to apply them with good intent.

Acknowledgements. This work is in part supported by Magic Grant from the Brown Institute for Media Innovation, the Samsung Global Research Outreach (GRO) Program, Autodesk, Amazon Web Services, and Stanford HAI for AWS Cloud Credits.

References

[1] Andreas Aristidou, Daniel Cohen-Or, Jessica K Hodgins, Yiorgos Chrysanthou, and Ariel Shamir. Deep motifs and motion signatures. *ACM TOG*, 2018. 2

[2] Guha Balakrishnan, Amy Zhao, Adrian V Dalca, Fredo Durand, and John Guttag. Synthesizing images of humans in unseen poses. In *CVPR*, 2018. 6

[3] Randy Bullock. Least-squares circle fit. *Developmental Testbed Center*, 2006. 3

[4] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE TPAMI*, 2019. 3

[5] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *ICCV*, 2019. 8

[6] Abe Davis and Maneesh Agrawala. Visual rhythm and beat. *ACM TOG*, 2018. 2

[7] Emily Denton and Rob Fergus. Stochastic Video Generation with a Learned Prior. In *ICML*, 2018. 2

[8] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to Infer Graphics Programs from Hand-Drawn Images. In *NeurIPS*, 2018. 2

[9] Hao-Shu Fang, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. RMPE: Regional multi-person pose estimation. In *ICCV*, 2017. 3

[10] Michael Gleicher. Motion editing with spacetime constraints. In *I3D*, 1997. 2

[11] Michael Gleicher and Peter Litwinowicz. Constraint-based motion adaptation. *The journal of visualization and computer animation*, 1998. 2

[12] Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *CVPR*, 2018. 2

[13] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *SIGGRAPH*, 2008. 2

[14] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. GRASS: Generative Recursive Autoencoders for Shape Structures. *ACM TOG*, 2017. 2

[15] Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. GRAINS: Generative Recursive Autoencoders for INdoor Scenes. *ACM TOG*, 2019. 2

[16] Ruilong Li, Shan Yang, David A. Ross, and Angjoo Kanazawa. Learn to dance with aist++: Music conditioned 3d dance generation, 2021. 7

[17] Aristid Lindenmayer. Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs. *Journal of theoretical biology*, 1968. 2

[18] Yu-Lun Liu, Yi-Tung Liao, Yen-Yu Lin, and Yung-Yu Chuang. Deep video frame interpolation using cyclic frame generation. In *AAAI*, volume 33, 2019. 2

[19] Ziwei Liu, Raymond Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video Frame Synthesis Using Deep Voxel Flow. In *ICCV*, 2017. 2

[20] Erika Lu, Forrester Cole, Tali Dekel, Weidi Xie, Andrew Zisserman, David Salesin, William T Freeman, and Michael Rubinstein. Layered neural rendering for retiming people in video. *SIGGRAPH Asia*, 2020. 2

[21] Jiayuan Mao, Xiuming Zhang, Yikai Li, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Program-Guided Image Manipulators. In *ICCV*, 2019. 2

[22] William McNally, Kanav Vats, Tyler Pinto, Chris Dulhanty, John McPhee, and Alexander Wong. Golfdb: A video database for golf swing sequencing. In *CVPR Workshop*, June 2019. 5

[23] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive separable convolution. In *CVPR*, 2017. 1, 2, 6, 7

[24] Chengjie Niu, Jun Li, and Kai Xu. Im2Struct: Recovering 3d Shape Structure from a Single Rgb Image. In *CVPR*, 2018. 2

[25] Charles F Rose, Bobby Bodenheimer, and Michael F Cohen. *Verbs and adverbs: Multidimensional motion interpolation using radial basis functions*. Citeseer, 1999. 2

[26] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: Neural Shape Parser for Constructive Solid Geometry. In *CVPR*, 2018. 2

[27] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to Infer and Execute 3D Shape Programs. In *ICLR*, 2019. 2

[28] Shuhei Tsuchida, Satoru Fukayama, Masahiro Hamasaki, and Masataka Goto. Aist dance video database: Multi-genre, multi-dancer, and multi-camera database for dance information processing. In *ISMIR*, pages 501–510, Delft, Netherlands, Nov. 2019. 7

[29] Ruben Villegas, Jimei Yang, Yuliang Zou, Sungryull Sohn, Xunyu Lin, and Honglak Lee. Learning to Generate Long-term Future via Hierarchical Prediction. In *ICML*, 2017. 1, 2, 8

[30] Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiquan Cheng, and Yueshan Xiong. Symmetry Hierarchy of Man-Made Objects. *CGF*, 2011. 2

[31] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE TIP*, 2004. 6

[32] Mao Wei, Liu Miaomiao, and Salzemann Mathieu. History repeats itself: Human motion prediction via motion attention. In *ECCV*, 2020. 1, 8

[33] Halley Young, Osbert Bastani, and Mayur Naik. Learning Neurosymbolic Generative Models via Program Synthesis. In *ICML*, 2019. 2

[34] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Networks As a Perceptual Metric. In *CVPR*, 2018. 6