SYNTHESIZING HIGH-QUALITY AND CONTROLLABLE TENNIS ANIMATION FROM REAL-WORLD VIDEO COLLECTIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Haotian Zhang
August 2023
© 2023 by Haotian Zhang. All Rights Reserved.
Re-distributed by Stanford University under license with the author.

This work is licensed under a Creative Commons Attribution-Noncommercial 3.0 United States License.
http://creativecommons.org/licenses/by-nc/3.0/us/

This dissertation is online at: https://purl.stanford.edu/nm925nc7955
I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Kayvon Fatahalian, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Ron Fedkiw

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Karen Liu

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format.
Abstract

Demonstrations of human performance are important for creating realistic virtual characters. However, collecting high-quality demonstrations via motion capture can be costly in both time and monetary cost, because it requires expensive environment setup and skilled performers to be present at the capture location. In contrast, there is an abundance of video capturing expert-level human performances. Given this observation, this thesis seeks to answer the question: how can we harvest information from internet videos to create realistic virtual human characters? In particular, online videos of athletic events provide a rich sampling of in-activity motion data which encompass the full spectrum of skills an athlete must perform in a sport. The challenge is that compared to the high quality of motion capture data, demonstrations extracted from monocular real-world videos will exhibit errors due to limitations of current machine perception algorithms, and source video deficiencies like occlusions and motion blur. This thesis centers its focus on a specific sports domain – tennis, and demonstrates that it is possible to use large-scale observations of athlete performance obtained from real-world video collections to create controllable, high-quality tennis animations of virtual characters playing singles tennis points. These characters successfully conduct tennis rallies, carry out realistic decision-making, and appear photorealistic.

Specifically, this thesis shows that a fairly simple state machine designed given the domain knowledge of tennis, is sufficient to provide structure for building complicated tennis controllers from unstructured video demonstrations. Next, this thesis contributes a system for learning a motion controller from real-world video demonstrations that is capable of precisely controlling a physically simulated character to play tennis points using a diverse set of skills. Finally, this thesis shows that macro-level behavior and player appearance information can be extracted from videos to enhance the realism of synthesized virtual characters. Overall, this thesis creates realistic virtual characters that can be effectively controlled to play singles tennis points involving a diverse array of strokes (serves, forehands, and backhands), spins (topspins and slices), and playing styles (one hand vs. two hand, left hand vs. right hand), as well as macro level behavioral choices that reflect player specific styles, such as shot selection choices. In the limited case of 2D sprite animation generation, it also demonstrates that appearance data from the videos can be employed to yield characters that are photorealistic in their appearance.
Acknowledgments

First, I would like to thank my advisor Kayvon Fatahalian. I was fortunate to become one of the first students to join his lab at Stanford. An interesting fact about this thesis is that I have never played tennis before. However, it is our common interest in large-scale video analysis, my passion for creating unprecedented graphics application and Kayvon’s expertise in tennis, that all together shaped this thesis. Over the past five years and more, I am extremely grateful for the dedicated time he invested in guiding me how to think and talk like a graphics researcher. I am also grateful for my undergraduate’s advisor Shi-Min Hu, who led me into the world of graphics and inspired me to pursue a career in research.

I would also like to thank Ron Fedkiw, Karen Liu, Jiajun Wu and Scott Delp for serving on my orals committee. During my PhD, I had the good fortunate to work and learn from a wonderful group of people at Stanford: James Hong (James deserves special thanks for collaborating with me through many projects and I am always amazed by his ideas and stories.), Cristobal Sciutto, Will Crichton, Fait Poms, David Durst, Dan Fu, Xinwei Yao, Brennan Shacklett, Vishnu Sarukkai, Purvi Goel, Jingwei Huang, Li Yi, Ning Jin, Zhenglin Geng, Maneesh Agrawala, Leo Guibas. I am also grateful for the opportunity to intern at NVIDIA and Adobe, working with talented and supportive teams: Ye Yuan, Xue Bin Peng, Viktor Makoviychuk, Yunrong Guo, Sanja Fidler, Long Mai (another special thank you for being my mentor throughout the intern project and beyond), Hailin Jin, Zhaowen Wang, Ning Xu, John Colomosse. Finally, I’d like to thank my friends Qiaoyi Liu and Jiaxuan You. We came from Tsinghua to Stanford at the same year and have been fitness buddy throughout my PhD. I really appreciate their support both physically and mentally.

To Zhixiu Liu, thank you for your patience and care over the past thousand day and nights. I could not imagine how I can come through this incredible journey without you being by my side.

Finally, I want to thank my parents, Ying Lu and Genying Zhang, for their support and care throughout my life. You always support and encourage me to learn and work on anything I enjoy, from drawing sketch as a little kid, to committing myself to pursue the PhD.
## Contents

Abstract iv

Acknowledgments v

1 Introduction 1

1.1 Key Challenges 3

1.2 Thesis Contributions 4

2 Modeling Tennis Points 6

2.1 Modeling Tennis Rally Structure 6

2.2 Modeling Tennis Ball Flight 8

2.3 Synthesizing A Tennis Point 9

3 Annotating Broadcast Videos 11

3.1 Dataset 11

3.2 Kinematic Motion Estimation 11

3.2.1 Player Tracking and Pose 12

3.2.2 Root Orientation and Trajectory 13

3.3 Physics-Based Motion Correction 13

3.3.1 States 14

3.3.2 Actions 14

3.3.3 Rewards 15

3.3.4 Training 16

3.3.5 Results and Discussion 17

3.4 Manual Annotations 17

3.4.1 Shot Cycle Boundaries 18

3.4.2 Shot Types and Outcomes 18

3.5 Ball Trajectory Estimation 18
## Tennis Control

4.1 Problem .................................................. 20
4.2 Related Work ........................................... 21
  4.2.1 Kinematic Control ................................. 22
  4.2.2 Physics-Based Control ........................... 23
4.3 A Simple Kinematic Controller ......................... 25
  4.3.1 Overview ........................................... 25
  4.3.2 Making Ball Contact ............................... 26
  4.3.3 Meeting Recovery Constraints ..................... 28
  4.3.4 Clip Cost .......................................... 28
  4.3.5 Point Ending ...................................... 30
  4.3.6 Results And Discussion ........................... 32

## Learning Physics-Based Tennis Control

5.1 Overview .................................................. 34
5.2 Motion Embedding ........................................ 36
  5.2.1 Pose Representation ............................... 36
  5.2.2 Motion Phase ...................................... 37
  5.2.3 Training ............................................ 38
5.3 High-Level Motion Planning Policy ...................... 38
  5.3.1 Policy Representation ............................. 39
  5.3.2 Reward Function .................................. 40
  5.3.3 Training ........................................... 40
  5.3.4 Additional Kinematic Constraints ................ 41
5.4 Results and Evaluation .................................. 42
  5.4.1 Experimental Setup ................................. 42
  5.4.2 Metrics ............................................. 43
  5.4.3 Learning Complex Tennis Skills ................. 43
  5.4.4 Tackling Low-Quality Demonstrations .......... 45
  5.4.5 Analysis of One Million Simulated Shots ....... 47
  5.4.6 Ablation Studies .................................. 48
  5.4.7 Sensitivity to Database Size ..................... 49
  5.4.8 Inspection of Joint Torques ....................... 50
  5.4.9 Effects of Removing Residual Force Control ..... 50
  5.4.10 Limitations ....................................... 52
6 Improving Fine-Grained Motion Details

6.1 Approach

6.1.1 Video Annotation

6.1.2 Wrist Optimization

6.1.3 Motion Blending

6.2 Results and Evaluation

6.2.1 Qualitative Evaluation

6.2.2 Quantitative Evaluation

7 Modeling Player Behaviors

7.1 Related Work

7.2 Approach

7.2.1 Shot Selection

7.2.2 Player Recovery Position

7.3 Results and Evaluation

7.3.1 Emergent Player Behavior

7.3.2 User Study

8 Synthesizing Photo-Realistic Appearances

8.1 Related Work

8.2 Video Sprite based Animations

8.2.1 Player Sprite Rendering

8.2.2 Appearance Normalization and Body Completion

8.2.3 Results and Evaluation

8.3 Neural Appearance Transfer

8.3.1 Approach

8.3.2 Results and Discussion

9 Conclusions

9.1 Toward Foundation Models for Human Motion

9.2 The Role of Simulation

9.3 The Role of Traditional Graphics Models

A Implementation Details and Additional Results
List of Tables

3.1 US Open dataset and Wimbledon dataset. We retain only video frames from the main broadcast camera, and discard all frames featuring instant replays or alternative camera angles. ........................................... 12

3.2 We utilize automatic machine annotations to estimate players’ poses and cameras at every frame and manual annotations for per-shot labels. .......................... 12

4.1 Threshold and weights used for each cost term. ........................................ 30

5.1 Task performance of controllers learned from three players’ motions using our system. We show the 25%, 50%, and 75% quantiles using the metrics collected from 10K test sessions. (The controller is tasked to hit 15 incoming balls in a drill-like setting per session.) The learned controllers consistently hit a high fraction of balls back into the court, and achieve average bounce position errors of less than two meters. .... 45

5.2 Ablations on the effect of physics correction (PhysicsCorr) and hybrid control (Hybrid-Ctr). Removing either component of the system results in decreased task performance. 45

5.3 Motion quality evaluation. We compare the motion output by our full system (Fed-full), the motion from the ablation (w/o PhysicsCorr), and motions at different stages of our system: estimated kinematic motion (M_{kin}), physically corrected motion (M_{corr}), and motion output by MVAE (M_{vae}). The motion generated from our full system shows higher motion quality (less jitter and foot sliding) than w/o PhysicsCorr and the motions at intermediate stages. ................................. 46

5.4 Ablation of various design choices of our system. The table provides average metrics collected from 10K test sessions. All design decisions contribute to the task performance of the controller. ........................................ 48

5.5 Removing residual force control yields more realistic motion (40% reduction in foot sliding) but reduces the tracking ability of the low-level imitation policy, resulting in reduced overall task performance. Users can select whether or not to employ residual forces based on desired performance-motion quality needs. .................. 51
5.6 Task performance of \textit{w/o RF} and \textit{Fed-full} when tested with a list of curated control problems with increased reaction distance (Y for successful hit and bounce-in). \textit{Fed-full} manages to move to reach the incoming balls when the reaction distance is large while \textit{w/o RF} fails. ......................................................... 51

6.1 Quantitative evaluation of improved style after integration of high-quality motions. The improved style also brings increase in both racket velocity and the shot velocity/spin after contact, which are closer to professional performance. ................. 59

8.1 Offline preprocessing times (both per frame, and for all frames (total), on a TITAN V GPU) for operations needed to preprocess database video clips to prepare them for rendering. .......................................................... 77

A.1 Hyper-parameters for training the low-level policy ................................. 84
A.2 Hyper-parameters for training the motion embedding model .................. 85
A.3 Hyper-parameters for training the high-level policy with the curriculum of three stages. 86
A.4 Discretization for each feature in the point descriptor. For the horizontal bins, “Ad/Center/Deuce” are split at $x = -1.4$ m and $x = 1.4$ m. For the vertical bins, “Front/Back” are split at $y = 9$ m. For the velocity to reach the contact point, we discretize the continuous velocity into five bins by $1.2$ m/s, $1.8$ m/s, $2.4$ m/s and $3.0$ m/s. ................................................................. 86
List of Figures

1.1 High-quality human demonstrations are typically captured in controlled motion capture studio settings for creating realistic virtual characters. (a) NBA2K (b) FIFA (c) Tennis World Tour. ................................................................. 2

2.1 A shot cycle for the player on the near side of the court. The cycle begins with the player in the “ready” state just as the opponent strikes the ball. Next, the player moves to hit the ball (“reaction” phase). He makes contact, striking the ball so that it lands on his opponent’s side of the court. After contact, the player returns to the ready position in a new court location to prepare for his opponent’s next shot (“recovery” phase). ................................................................. 7

2.2 Ball flight model in different spins. Topspin (forward ball rotation) imparts downward Magnus force to the ball while backspin (backward ball rotation) produces upward Magnus force. Air drag is always opposite to the direction of the ball’s velocity. 8

2.3 Sketch of the pipeline for synthesizing a tennis point. ........................................... 9

3.1 Example of per-frame machine annotations. (a) Source frame. (b) 2D annotations including: player bounding boxes, 2D pose keypoints, foot contact detections and court line detections. (c) 3D pose (SMPL mesh) and global root positions . . . . . . 13

3.2 Overview of the imitation policy. Given the current state of the character and a reference motion, the policy specifies the PD targets and residual forces/torques that allow the simulated character to track the reference motion as closely as possible. . 14

3.3 Examples of the corrected motion. Top: the simulated character (pink) is able to track the kinematic motion (yellow) closely when controlled by the imitation policy. Bottom: the corrected motion is still inaccurate (wrist and neck motion) when compared to the original video. Note in the leftmost column, the character has a locked wrist and the head is not pointing toward the ball. In the rightmost column, the character stops the racket in the air while the player has the racket stopped over his shoulder. ................................................................. 17
3.4 Given the space-time location (time, 3D position) of consecutive ball contacts during a rally, we find a ball trajectory that starts at the first space-time contact location, clears the net, bounces in the court, and then closely matches the ending location. We use grid-search over the ball’s initial linear velocity (horizontal and vertical components) and spin velocity to find a matching trajectory.  

3.5 Estimated trajectories in database clips for all players separated by forehand/backhand shots. The red line indicates the mean of trajectories’ maximum height before bounce. Nadal has the highest trajectories among the three male players. 

4.1 Input to the tennis control problem. 

4.2 Translational correction in clip search. Using motion directly from a database clip would place the player’s racket at $x_c$ at ball contact time, and the player at $x_r$ at the end of the shot cycle. We compute the error in these positions $e_c$ (the distance from $x_c$ to the ball) and $e_r$ (distance to the target recovery position $x_r^*$), and apply a translational correction to the root position of the player so that the racket meets the incoming ball, and that player reaches the target recovery position at the end of the shot cycle. 

4.3 Results of the simple controller. (a) The character starts with a ready pose at the beginning of the shot cycle (b) The character hits the ball along the incoming ball trajectory with target shot type (c) The outgoing ball matches the target shot spin (d) The outgoing ball bounces at the target shot placement position (e) The character recovers to the target recovery position at the end of the shot cycle. 

4.4 Clip search cost (averaged over 400 simulated rallies) decreases most quickly for Federer, suggesting that his controlled style of play yields a narrow distribution of player configurations. Except for $C_{\text{react.dir}}$ (red line), Nadal’s simulations have higher costs than those for Federer and Djokovic. The plots normalize each cost term by its maximum value encountered over all experiments (values across graphs are comparable). 

4.5 The statistics of how frequently database clips are used during the synthesis of 1500 randomly generated points. Almost 90% of the database clips are used at least once. 

5.1 We present a hierarchical physics-based controller that allows simulated characters to learn tennis skills from motions harvested from broadcast tennis videos. Our simulated characters can hit the incoming ball to target positions accurately using a diverse array of strokes (serves, forehands, and backhands), spins (topspins and slices), and playing styles (one/two-handed backhands, left/right-handed play). Overall, our system can synthesize two physically simulated characters playing extended tennis rallies with simulated racket and ball dynamics.
5.2 Overview of the hierarchical controller. We fit a conditional VAE to the corrected
motion to learn a motion embedding that produces diverse and human-like tennis
motions. A high-level motion planning policy is then trained to generate target kine-
matic motion by predicting VAE latent codes and joint corrections for wrist motion.
Finally, the target motion is imitated by the low-level imitation policy to control a
physically simulated character to perform the desired task. ......................... 36

5.3 Overview of the motion embedding model. At training time, given the character’s
current pose and a latent variable \( z \), the model reconstructs the pose in the next time
step and outputs the motion phase while shaping \( z \) into a normal distribution. At
test time, the encoder is discarded and the decoder takes the input of the current
pose and a latent \( z \) to produce the next pose and motion phase. .................... 37

5.4 Overview of the high-level planning policy. The high-level motion planning policy
is trained to generate target kinematic motion by combining body motion from the
motion embedding with predicted corrections for the character’s wrist motion. The
target motion is then imitated by the low-level imitation policy to control a physically
simulated character to perform the desired task. ......................... 39

5.5 The simulated character model is created from SMPL [53], with 24 rigid body seg-
mants and 72 DOF. The tennis racket is a combination of two solid cylinders and the
grip is simplified by directly attaching the end of the racket handle to the wrist joint. 42

5.6 Our simulated characters demonstrate diverse tennis skills that reflect coarse char-
acteristics of the per-player video data they were trained on (a)-(d) skills learned
using Roger Federer’s motion data, who is a right-handed player and uses one-handed
backhand. (e)-(f) skills learned using Novak Djokovic’s motion data, who is also a
right-handed player but uses two-handed backhand. (g)-(h) skills learned using Rafael
Nadal’s motion data, who is a left-handed player and uses two-handed backhand. . . 44

5.7 Analysis of 1M simulated shots. (a) 2D heat maps of hit rate conditioned on incoming
velocity and amount of spin. Balls with higher velocity and spin are harder to hit.
(b)-(c) 2D heat maps of hit rate and average bounce position error conditioned on the
incoming ball’s bounce position. Balls that bounce closer to the edges of the court
are harder to hit and result in higher bounce position errors. (d)-(e) Moving longer
distances to reach an incoming ball (reaction distance) results in lower hit rates and
larger bounce position errors since the character must move quickly and has less time
to adjust. ................................. 47

5.8 Larger video database sizes (more motion) yield controllers with increased task per-
formance (higher hit rate and bounce-in rate) and improved motion quality (lower
jitter). ........................................ 49

5.9 Box plot of joint torques generated by the PD controller when controlled by Fed-full. 50
5.10 Root velocity of the simulated characters under different reaction distances (RD), when controlled by Fed-full (left) and w/o RF (right). The simulated character runs faster with residual force control enabled. The maximum root velocity is above 5 m/s when controlled by Fed-full, while it is below 4 m/s when controlled by w/o RF.

5.11 Residual forces (left) and torques (right) under different reaction distances when the character is controlled by Fed-full. The required residual forces/torques are small when the characters do not need to run very fast, while they quickly explode when tracking fast running reference motions.

5.12 Failure cases. Top: the character simulated without residual force control may fall down when tracking fast running reference motion. Bottom: the character simulated with residual force control may exhibit implausible balance when running fast to reach the ball and trying to recover from the swing motion.

6.1 The learned controller fails to reproduce the fine-grained motion details from the original performance. Left: the character does not exhibit clear pronation during back swing; Right: the follow-through is not long enough to slow down the fast swinging racket.

6.2 Examples of slow-motion videos for Federer’s forehand swings.

6.3 Comparison between reconstructed motions from broadcast videos (top) and slow-motion videos (bottom). The yellow and pink characters visualize the kinematic estimated motion and the simulated motion from the low-level imitation policy. Note that the fine-grained motion details such as pronation (first column) and longer follow-through (last column) are neither captured in the kinematic motion or the simulated motion when reconstructed from broadcast videos. However, these details are better preserved when reconstructed from slow-motion videos.

6.4 Comparison between motions generated by controllers learned using the old motion embedding (top) and the corrected motion embedding (bottom). The motion generated by Fed-HQ shows more clear wrist pronation before contact (a), more wrist bend during the swing (b)(c), and a longer follow-through after contact (d).
7.1 We build player behavior models using player and ball trajectories extracted from video to predict a player’s shot placement (where they hit the ball to) and where they position themselves on the court against a specific opponent, which can be used as the input constraints for the tennis control problem. As a result, the synthesized tennis points share similar style and strategy as their real-world counterparts. Visualizations depict where our model of Novak Djokovic hits the ball to (shot placement) and positions himself (recovery position) when hitting backhands against Roger Federer and Rafael Nadal. Djokovic’s strategy changes to hit to the right side of the court vs. Federer (his weaker backhand side) while a majority of shots vs. Nadal go to the left.

7.2 Visualization of the shot placement distribution, \( p(x_b|D_s, c = FH-T) \) (red), and recovery position distribution, \( p(x_r|D_r, Y = 0) \) (blue), given a point configuration where the player’s estimated ball contact point lies within the cyan region, the opponent’s expected position at ball contact lies within the purple region, and the incoming ball follows the given trajectory (orange). (FH-T denotes a forehand-topspin shot, \( Y = 0 \) denotes the decision to not approach the net during post-shot recovery.) The dotted court boundaries denote the regions used to discretize the positions of players and the incoming ball’s starting and bounce positions. The velocity required for the player to reach the estimated contact point is also discretized to determine \( D_s \). Yellow dots indicate the shot placement positions and recovery positions for database clips that match \( D_s \) and \( D_r \), and are used to estimate the distributions.

7.3 Shot placement (forehand/backhand) and recovery positions of Federer (right-handed), Djokovic (right-handed), and Nadal (left-handed) in synthesized points. (The red and blue heat maps visualize a KDE fit to the results of simulation, not the distribution used by our behavior models.) In general a majority of shots are hit cross court, except for Nadal’s backhand which is distributed equally around the court. Federer stands nearly one meter closer to the baseline (only 0.9 m behind baseline) than Djokovic (1.6 m) and Nadal (1.9 m), suggesting more aggressive play. All players recover to an off-center court position, making it harder for their opponents to hit the ball to their backhand side.
7.4 Synthesized points feature emergent player behavior that is consistent with good tennis principles. (The red and blue heat maps visualize a KDE fit to the results of synthesized points, not the distribution used by our behavior models.) (a) When hitting the ball from left side of the court, virtual Nadal places most of his shot cross-court to Federer’s significantly weaker backhand side. When playing Djokovic, Nadal directs more shots down the line. (b) When virtual Djokovic has to move rapidly (to the right) to hit an incoming shot vs. Nadal, he plays defensively, placing most of his shots cross court even though that is where Nadal is standing. When Djokovic does not need to move quickly to reach the incoming ball, he plays more aggressively, placing a greater fraction of shots down the line to the open court. (c) Most of the time when virtual Federer approaches the net is after hitting a short incoming ball down the line to the open court. Positioning oneself at the net following a down the line shot is a common attacking strategy in tennis.

7.5 Participants rated our shot selection behavior model ($\mu = 3.84, \sigma = 0.83$) better than no behavior model ($\mu = 3.2, \sigma = 0.89$) but less realistic than the oracle ($\mu = 4.48, \sigma = 0.70$). We find all of these differences to be significant running Friedman’s non-parametric test ($p < 0.001, \chi^2 = 24.33$) followed by pairwise comparisons using Wilcoxon’s signed rank tests ($p < 0.019, Z = 44.50$ ours vs. no behavior, $p < 0.005, Z = 24.00$ ours vs. oracle). Similarly participants rated our recovery behavior model ($\mu = 3.76, \sigma = 0.76$) better than no behavior model ($\mu = 2.16, \sigma = 1.12$) but less realistic than the oracle ($\mu = 4.84, \sigma = 0.37$). Again we find all these differences to be significant running Friedman’s test ($p < 0.001, \chi^2 = 37.42$) followed by pairwise comparisons using Wilcoxon’s tests ($p < 0.001, Z = 19.50$ ours vs. no behavior, $p < 0.001, Z = 0.00$ ours vs. oracle).

8.1 Synthesized tennis videos of novel points. (a) Federer (near) vs. Federer (far) (b) Williams (near) vs. Federer (far) (c) Nadal (near) vs. Federer (far) (d) Williams (near) vs. Gauff (far). Note none of the match-up has ever happened in real life.

8.2 Source video sprites are translated and scaled to render the player at a novel location in court space. Scaling requires an estimate of the court space extent of the player’s bounding box, which is computed from projection of the boxes’ bottom-left and bottom-right screen space points ($\tilde{x}_A, \tilde{x}_B$).

8.3 When the player in an input clip lies partially outside the frame, we hallucinate missing pose keypoints from similar (fully visible) sprites, then use paired image translation to hallucinate missing pixels from the completed skeleton. In this example, the final sprite contains a combination of original pixels and hallucinated legs.
8.4 We train CycleGAN models [111] to perform unpaired image-to-image translation to reduce differences in a player’s appearance across video clips. These corrections enable a larger number of video clips to be used as inputs for video sprite synthesis. Left: removing lighting differences. Right: modifying Serena Williams’ clothing (adding long sleeves) to match her outfit on a prior day. 

8.5 Input and output of our neural appearance transfer model. (a) Input empty background (b) Input 2D skeletal figure (c) Output image. 

8.6 Synthesizing novel tennis point using the neural appearance transfer model. (a) Input simulated 3D motion (b) Projected 2D skeletal figure onto US Open background (c) Synthesized point in US Open background (d) Synthesized point in Wimbledon background. 

A.1 Shot placement (forehand/backhand) and recovery positions of Federer (right-handed), Djokovic (right-handed), and Nadal (left-handed) in our simulated points (top) and database points (bottom). By modeling behaviors from real-world data, we can simulate points that capture player-specific styles that are consistent with well-known tendencies of the real-life players. 

A.2 Image-based approach for player shadow rendering. We warp the player’s alpha matte using three control points (red dots) from (a) to (c) to approximate projection on the ground plane from a user-specified lighting direction. We warp the matte ten times around the give lighting direction to simulate soft shadow (d) and darken the pixels in the warped matte to render the final player shadow. 

A.3 Montage of appearance normalization results. The top row shows the montage of uniformly sampled cropped frames around nearside players throughout each source video (ordered by day time). The bottom row shows the montage of the corresponding frames with appearance normalization, which are processed to share the same lighting conditions. 

A.4 Appearance normalization results for relighting the male players (top) and relighting and re-clothing Serena Williams (bottom). The normalized sprite is shown in the black box and the original sprite is shown to its left. Notice how normalization corrects for significant sunlight differences that occur over men’s matches that take place outdoors over the span of hours, and adds long sleeves to William’s shirt. (high-resolution image—please zoom in) 

A.5 Body completion results for Nadal (top) and Djokovic (bottom). The completed sprite is shown in the black background and the original cropped sprite is shown to its left. (high-resolution image—please zoom in)
Chapter 1

Introduction

Creating realistic virtual human characters has long been a goal of computer graphics, with broad implications to areas such as real-time games, cinematic effects and robotics. Today, most systems that produce realistic human animations are based, in some way, on mimicking recorded examples of human movement. The challenge of this data-driven approach is that creating high-quality output depends heavily on access to high-quality capture of human performance. For this reason human demonstrations are typically captured in controlled motion capture studio settings (Figure 1.1). High-quality capture requires expensive environment setup and skilled performers to be present at the capture location. As a result, data collection is expensive in both time and monetary cost. Compared to the large collections of internet images, videos, and text that drive data-driven methods in other areas of computer graphics today, there is relatively little high-quality human motion data available.

However, massive scale demonstrations of human performance, in particular by expert performers, are widely available in the form of online videos. For example, let’s consider the domain of sports. From high-school games to professional leagues, it is common for athletic competitions to be captured in video. These widely available videos offer a wealth of information about how skilled athletes move (low-level movement), look (appearance, style of motion) and play their sport (tendencies, strengths, and weaknesses). Compared to what demonstrations might be captured in a motion capture studio, online videos of athletic events provide a rich sampling of in-activity motion data. Full game videos encompass the full spectrum of skills an athlete must perform in a sport: not just salient actions (hitting a shot in tennis), but the complex and subtle motions athletes use to transition between these movements. Further, long-running videos allow for observation of many variations of each action (hitting a high ball or a low ball in tennis, or reacting quickly or slowly to a shot by an opponent).

The challenge is that these data sources are real-world videos (monocular, modest resolution, lower frame rate), not high accuracy performance capture data from a motion capture lab. Even
CHAPTER 1. INTRODUCTION

Figure 1.1: High-quality human demonstrations are typically captured in controlled motion capture studio settings for creating realistic virtual characters. (a) NBA2K (b) FIFA (c) Tennis World Tour.

though modern computer vision methods make it possible to automatically detect and track players across frames, estimate their poses in 2D and 3D, and derive global positional information about their motion, these outputs remain significantly lower quality than professional motion capture results. Moreover, generating realistic characters involves more than just reconstructing fine-grained motion: virtual characters must exhibit realistic longer-term behaviors, or respond realistically to user-provided control inputs. Therefore, the central question of this thesis is this: how can we harvest information about human performance from widely available monocular internet videos and operationalize it to create high quality, controllable virtual human characters?

Attempting to address this question across all domains would be ambitious. Instead, this thesis centers its focus on a specific sports domain – creating characters that can play singles tennis points. I chose singles tennis because it seemed sufficiently challenging, but also tractable. Compared to other multiplayer team sports, singles tennis poses an easier perception problem: it features only two visible players, instances of occlusion are minimal, and in contemporary tennis broadcasts the camera position is usually stationary, with only small changes to its rotation and zoom over time. Still, as anyone that has attempted to play tennis in a local park knows, playing tennis points is a challenging activity. Tennis features dynamic human motion, requires a diverse range of skills and complex motion transitions between these skills, and has a high bar for motion precision: small differences in a player’s swing can lead to very different shot outcomes.

Through the work in this dissertation, I aim to show that it is possible to use large-scale observations of athlete performance obtained from real-world video collections to create controllable, high-quality tennis animations of virtual characters playing singles tennis points. To successfully utilize large amounts of unstructured video data to create virtual characters, my solutions draw upon constraints provided by physics and human knowledge of the structure of tennis. Specifically, by **controllable** I seek both “skills-level” control: the ability for characters to successfully conduct rallies by returning the incoming tennis balls to the court, but also “behavior-level” control, as carrying out realistic decision-making given the current state of the point (e.g., where to place the next shot). By **high-quality** I mean that the generated character motions should
resemble human play, and that entire rallies should be “plausible” in their overall structure. In the limited case of 2D sprite animation generation, I also demonstrate how to harvest appearance data from the videos to yield characters that are photorealistic in their appearance.

1.1 Key Challenges

We organize the challenges and opportunities that arise when constructing animation systems from real-world tennis videos into the following aspects:

**How to model the fundamental structure of tennis rallies?** Tennis is a complicated activity, which consists of a variety of skills and natural transitions between these actions. Developing a controller for playing tennis requires both high-level planning (choosing when and where to hit the ball) and low-level execution (moving to the desired location and hitting the ball). Given only untrimmed tennis videos, it is challenging to recover the fundamental structure of tennis rallies. Since modeling this structure seems useful for both interpreting source videos and generating plausible rallies, an interesting question is what is the basic structure of tennis we can author, and what is the minimum domain knowledge we can employ in order to regularize data-driven techniques.

**How to automatically extract tennis motion and point play information from monocular broadcast videos that are of sufficiently high quality for use in creating virtual characters?** The input data we used is unannotated and untrimmed broadcast tennis videos with abundant information. Creating virtual characters using these videos requires a nearly automatic pipeline to annotate not only the per-frame annotations such as player pose and positions, but also per-shot annotations based on the tennis structure, e.g. the time when players make ball contacts. Furthermore, the fundamental challenge lies in the low-quality motion harvested from broadcast videos (monocular, modest resolution, lower frame rate). Employing off-the-shelf vision based motion estimation will lead to physically implausible motion, characterized by artifacts such as jitter, foot skating and ground penetration. More importantly, wrist motion for controlling the tennis racket can not be reliably estimated due to occlusion and motion blur, which requires additional solution to hallucinate the missing motion when developing the controller.

**How can we learn a controller to master a diverse range of tennis skills from only watching tennis points?** First, our goal is to build a single controller that can produce diverse tennis skills (different strokes and spins) and the transitions between these skills to play a full tennis rally. Second, playing tennis requires precise spatio-temporal control of the human movement. Even the slightest deviation, such as missing the racket-ball contact by a few centimeters in space or sub-seconds in time, is considered a failure. This stringent requirement for spatial-temporal accuracy demands highly precise control strategies. Finally, while there is an abundance of tennis motion...
data available from videos, the availability of ground-truth control problems and their corresponding solutions (i.e., player motion paired with the incoming ball trajectory) is sparse. In a typical tennis match, we can only extract a few hundreds of shots with such paired data. This scarcity of accurately annotated control problems makes it challenging to develop effective solutions based on supervised learning.

**Can we extract higher level strategy (behavior) from the video?** Creating high-quality virtual characters is more than just making them move realistically, but also have them make reasonable behavior choices. Such information can actually be sourced from videos as well. For instance, long match videos provide rich logs of how a player makes decisions, offering an opportunity to model a player’s macro level behaviors during point play, such as real-life strategies (hitting the ball to an opponent’s weakness) and tendencies (aggressive vs. defensive court positioning), akin to the field of sports analytics. Therefore, the question is can we extract higher level strategy (behavior) from the video to make the generated tennis rally realistic at a point level?

**Can we extract appearance information from the video?** Furthermore, it might be difficult to create the exact geometry and materials of the players for use in a graphics rendering pipeline for visualizing the animation. However, the pixels from the source video offer abundant information about the player appearances, which can be potentially used when rendering the animation from the same camera views, via video textures or image based transferring techniques. However, real-world footage presents challenges such as differences in player appearance over different match days and times and missing pixel data when players are partially cropped in the frame. Therefore, the question is can we extract appearance information from the video to create photorealistic tennis animations?

### 1.2 Thesis Contributions

In addressing the challenges and embracing the opportunities associated with real-world videos for synthesizing high-quality and controllable tennis animations, this thesis makes the following contributions:

**Modeling tennis point structure as a sequence of shot cycles.** First, this thesis shows that a fairly simple state machine designed given the domain knowledge of tennis, is sufficient to provide structure for building complicated tennis controllers. Throughout this thesis, we show that this tennis specific state machine can provide structure for aiding data annotation and specifying control inputs. It can also effectively regularize the data-driven character controllers, such as set the granularity of the control problem and simplify the learning of the control policies.
Learning a motion controller from real-world video demonstrations that is capable of precisely controlling a physically simulated character to play tennis points with diverse skills. Second, this thesis demonstrates that low-quality motions estimated from large-scale broadcast videos can be leveraged for building high-fidelity tennis controllers that enable diverse and precise control. Specifically, a nearly automatic pipeline is built for extracting motion data from monocular videos where physically implausible motions estimated from videos can be corrected via physics-based imitation. The corrected motions then can be embedded into a motion model, which serves as the foundation for building a single hierarchical controller to manage diverse skills via reinforcement learning. Finally, erroneous joint motions that persist in the learned motion space (e.g., wrist motion) can be overridden using physics-based corrections driven by high-level task rewards to enable precise control. Additionally, we demonstrate that sparse but high fidelity motion examples from close-up, slow-motion recordings, can be effectively integrated into the motion space, so that the final simulated skills better resemble those of the original performance.

Demonstration that macro-level behavior and player appearance information can be extracted from broadcast videos to enhance the realism of synthesized virtual characters. Finally, this thesis shows that additional information from the source videos, such as behavior and appearance, can be exploited to make the virtual characters not only move realistically, but also look and behave like professional tennis players. First, player-specific behavior models can be constructed by extracting historical play data from long match videos. By utilizing these behavior models, we can generate animations that are “realistic” at a macro level in that they capture real-life strategies and tendencies, and leads to more realistic points as judged by expert tennis players. Moreover, player appearances from the source video can be directly incorporated into a video-texture based animation system to synthesize videos with photo-realistic appearances. Alternatively, they can serve as training data for neural appearance transfer models to synthesize realistic appearances conditioned on generated motion.
Chapter 2

Modeling Tennis Points

Since the goal of the thesis is to produce animations of virtual characters playing tennis points, we need to model and simulate tennis points that effectively portray both the players and the ball in a compelling manner. In this chapter, we will present how we model tennis points, encompassing both the structure of how players make decisions and move during the point, and how the ball flights as a result of their actions. First, we will demonstrate that the structure of tennis rallies can be effectively modeled using a simple state machine, which we refer to as the shot-cycle state machine (Section 2.1). The key elements of the shot-cycle state machine will be described, including the states and control inputs, which also defines the tennis control problem we are aimed to solve. This is the essential structure that will be employed throughout the thesis, such as to aid the annotation of the broadcast tennis videos, and serve as the foundation for building the tennis controllers. Second, we will delve into physics modelling of the ball flight. The accuracy and realism of our simulated tennis point heavily rely on the faithful representation of ball flights, ensuring that they appear physically plausible. To achieve this, we will introduce the physics background required for simulating the tennis ball flight and describe the physics model we utilized in our simulation (Section 2.2). By integrating the shot-cycle state machine and the physics-based ball flight model, we will lay a solid foundation for modeling the tennis point, which also establishes the pipeline of synthesizing the animation of a full tennis point (Section 2.3). Following this pipeline, we organize the rest of the chapters as components to fill in the details of the pipeline.

2.1 Modeling Tennis Rally Structure

In tennis, play is broken into points, which consist of a sequence of shots (events where a player strikes the ball). A point begins with a serve and continues with a rally where opposing players alternate hitting shots that return the ball to their opponent’s side of the court. The point ends when a player fails to successfully return the incoming ball, either because they can not reach the
Figure 2.1: A shot cycle for the player on the near side of the court. The cycle begins with the player in the “ready” state just as the opponent strikes the ball. Next, the player moves to hit the ball (“reaction” phase). He makes contact, striking the ball so that it lands on his opponent’s side of the court. After contact, the player returns to the ready position in a new court location to prepare for his opponent’s next shot (“recovery” phase).

During a rally, players make a critical sequence of movements and decisions in the time between two consecutive shots by their opponent. We refer to this sequence, which involves reacting to the incoming ball, hitting a shot, and finally recovering to a position on the court that anticipates the next shot, as a shot cycle. Figure 2.1 illustrates one shot cycle for the player on the near side of the court.

Phase 1: (Ready→Reaction→Contact). At the start of the shot cycle (when the player on far side of the court in Figure 2.1 strikes the ball) the near player is posed in a stance that facilitates a quick reaction to the direction of the incoming shot. We mark this moment as ready on the timeline. Next, the near player reacts to the incoming ball by maneuvering himself to hit a shot. It is at this time that the player also makes shot selection decisions, which include: shot type, what type of shot to use (e.g., forehand or backhand, groundstroke or volley); shot spin, what type of spin to use (topspin or underspin); shot velocity, how fast to hit the shot; and shot placement position, where on the opponent’s side of the court they will direct their shot to. We refer to the period between ready and the time of ball contact as the reaction phase of the shot cycle.
CHAPTER 2. MODELING TENNIS POINTS

Figure 2.2: Ball flight model in different spins. Topspin (forward ball rotation) imparts downward Magnus force to the ball while backspin (backward ball rotation) produces upward Magnus force. Air drag is always opposite to the direction of the ball’s velocity.

Phase 2: (Contact→Recovery→Ready). After making contact and finishing the swing, the near player moves to reposition themself on the court in anticipation of the opponent’s next shot. This movement returns the player back to a new ready position right as the ball reaches the opponent. We call this new ready position the recovery position and the time between the player’s ball contact and arrival at the recovery position the recovery phase.

The shot cycle forms a simple state machine that governs each player’s behavior during a rally. In a rally, both players repeatedly execute new shot cycles, with the start of the reaction phase of one player’s shot cycle coinciding with the start of the recovery phase in the current shot cycle of their opponent. (The start of a shot cycle for one player is offset from that of their opponent by half a cycle.)

2.2 Modeling Tennis Ball Flight

In addition to the shot-cycle state machine which models the coarse structure of how players move during a rally, to simulate a tennis point the system also model the dynamics of how the ball flies when it is hit by players. The trajectory of a tennis ball depends significantly on air drag and on the Magnus forces resulting from the ball’s spin (shown in Figure 2.2). Specifically, topspin (forward ball rotation) imparts downward acceleration to the ball leading it to drop quickly. Backspin (backward ball rotation) produces upward acceleration causing the ball to float [7]. We adopt a physics-based ball flight model, where the tennis ball is simulated as a rigid sphere with the same radius and mass as a real tennis ball, with a restitution of 0.9 and friction of 0.8. To simulate air friction and the effects of spin, we can model air drag force $F_d$ and Magnus force $F_M$ as follows:

$$F_d = C_d A v^2 / 2, \quad F_M = C_L A v^2 / 2,$$

(2.1)

where $v$ denotes the magnitude of the ball’s velocity and $A = \pi \rho R^2$ is a constant determined by the air density $\rho = 1.21$ kg/m$^3$ and the ball’s radius $R = 0.032$ m. $F_d$ is always opposite to the
CHAPTER 2. MODELING TENNIS POINTS

2.3 Synthesizing A Tennis Point

By integrating the shot-cycle state machine and the physics-based ball flight model, we can establish the pipeline of synthesizing the animation of a full tennis point. Specifically, at the beginning of the \( n^{th} \) shot cycle \( C^p_n \) for the current player \( p \), the system generates behavior targets (motion goals) \( G^p_n \). These targets might include decisions like the shot type (forehand/backhand), spin (topspin/underspin) and shot placement position, as well as the player’s recovery position. Then a controller is required to solve the tennis control problem, which is to generate plausible animation \( M^p_n \) for the current player that follows the previous animation \( M^p_{n-1} \) and meets the behavior targets \( G^p_n \). When player \( p \) makes ball contact during the shot cycle \( C^p_n \), the ball flight model generates the ball trajectory \( B^p_n \) that results from \( p \)'s shot. Now, the inputs of the control problem for the opposing player \( o \)'s next shot cycle are known. The system proceeds by generating behavior targets for the opponent \( G^o_n \). This process continues iteratively until the end of the point. In summary, a full tennis point can be synthesized given (\( \ldots \ G^p_n, M^p_n, B^p_n, G^o_n, M^o_n, B^o_n, G^p_{n+1}, M^p_{n+1}, B^p_{n+1} \ldots \)) which alternates between the two players (Figure 2.3).

Following this point synthesis pipeline, we organize the rest of the chapters as components of this pipeline. We will proceed to explain how we annotate broadcast tennis videos to extract essential information for modeling tennis and organize the tennis dataset based on the shot-cycle
structure in Chapter 3. Next, we introduce a more detailed formulation of the control problem in Chapter 4, where we also review the literature of data-driven human controllers and present a simple kinematic controller designed for the control problem. We then describe our physics-based tennis controller which addresses the limitations of the simple controller in Chapter 5 and discuss solutions to improve the physics-based controller with additional data sources in Chapter 6. Finally, we discuss how to model player behaviors for predicting the behavior constraints in Chapter 7 and introduce solutions to employ the appearance information from the videos to enhance the photo-realism of the synthesized tennis animations in Chapter 8.
Chapter 3

Annotating Broadcast Videos

In this chapter, we describe the broadcast tennis videos we used and the full pipeline for annotating these videos into useful information for synthesizing controllable characters. To build our tennis dataset from the raw match videos, we utilize automatic machine annotations to estimate players’ kinematic motion (Section 3.2) and employ physics-based motion imitation to improve the physical plausibility of the annotated 3D motions (Section 3.3). We also conduct manual annotations for players’ identities and shot-specific labels, such as racket-ball contact times (Section 3.4).

3.1 Dataset

We collect 13 US Open match videos, which consist of matches of three players Roger Federer, Rafael Nadal, and Novak Djokovic playing against each other or against other players. We also collect 5 Wimbledon match videos, which consist of three matches of Federer, Nadal, Djokovic playing against each other along with two matches of Serena Williams playing against Simona Halep and Camila Giorgi. All the videos are recorded in 1080p with frame rate of 30 for all the US Open videos and 25 for all the Wimbledon videos. We retain only video frames from the main broadcast camera, and discard all frames featuring instant replays or alternative camera angles. We list the total amount of video/motion length for each player of the two datasets in Table 3.1. For instance, we are able to collect 80 minutes of motion for Federer out of 11 hours of US Open match videos. Table 3.2 provides a glossary for the symbols used to represent all quantities annotated in the videos.

3.2 Kinematic Motion Estimation

We use off-the-shelf detection models to annotate players’ kinematic motion, including player tracking, pose estimation, and global root orientation/trajectory estimation.
CHAPTER 3. ANNOTATING BROADCAST VIDEOS

<table>
<thead>
<tr>
<th></th>
<th>Number of matches</th>
<th>Length of videos (h)</th>
<th>Length of in-play clips (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>US Open dataset</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R. Federer</td>
<td>4</td>
<td>11</td>
<td>80</td>
</tr>
<tr>
<td>R. Nadal</td>
<td>5</td>
<td>14</td>
<td>103</td>
</tr>
<tr>
<td>N. Djokovic</td>
<td>5</td>
<td>14</td>
<td>96</td>
</tr>
<tr>
<td><strong>Wimbledon dataset</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R. Federer</td>
<td>2</td>
<td>8</td>
<td>58</td>
</tr>
<tr>
<td>R. Nadal</td>
<td>2</td>
<td>8</td>
<td>57</td>
</tr>
<tr>
<td>N. Djokovic</td>
<td>2</td>
<td>10</td>
<td>79</td>
</tr>
<tr>
<td>S. Williams</td>
<td>2</td>
<td>3</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 3.1: US Open dataset and Wimbledon dataset. We retain only video frames from the main broadcast camera, and discard all frames featuring instant replays or alternative camera angles.

**Schema for video annotations:**

For all frames (indexed by time $t$):

- $x_p(t)$: Player’s root position (court space)
- $o(t)$: Player’s root orientation
- $\bar{p}(t)$: Player’s 2D pose keypoint
- $q(t)$: Player’s 3D joint rotations
- $c_l(t), c_r(t)$: Player’s foot contact labels
- $H(t)$: Camera projection mapping court space to screen space

For each shot cycle (indexed by $i^{th}$ cycle):

- $t^i_c$: Time of player’s ball contact (length of reaction phase)
- $t^i_r$: Time of end of recovery phase (length of the clip)
- $c^i$: Shot type
- $o^i$: Shot outcome
- $v^i_b$: Shot velocity
- $x^i_c$: Shot contact position
- $s^i$: Shot spin
- $t^i_b$: Time of ball bounce on the court
- $x^i_b$: Shot placement position

Table 3.2: We utilize automatic machine annotations to estimate players’ poses and cameras at every frame and manual annotations for per-shot labels.

### 3.2.1 Player Tracking and Pose

We track the players and estimate their 2D/3D poses from the broadcast videos by using off-the-shelf detection models. We run Yolo4 [5] to track players on both sides of the court to obtain point boundaries and player bounding boxes. 2D pose keypoints ($\bar{p}(t)$) are extracted using ViTPose [99]. We also generate binary foot contact labels ($c_l(t), c_r(t)$) using the prediction model from Yu et al. [105]. Finally, we use HybrIK [44] to estimate the joint rotations ($q(t)$) for SMPL [53]. Player bounding boxes are used to crop the images around the player before being provided as input to the pose estimator.
3.2.2 Root Orientation and Trajectory

Since HybrIK only outputs root position and orientation in camera coordinates, those quantities need to be converted to the global court coordinates, where the origin is located at the center of the court. We first estimate the camera projection ($H(t)$) using the method from Farin et al. [17] to detect court lines and their intersections, and then solve for the camera matrix with the Perspective-N-Point algorithm. The camera transformation can then be used to obtain the global root orientation ($o(t)$). To estimate the player’s global root position ($x_p(t)$), instead of using the translation from the camera transformation, we first compute the 2D position of the player’s root projected onto the ground (center of two ankle keypoints), then transform the location into court coordinates with the inverse camera projection. We further correct the root trajectory by solving an optimization problem similar to GLAMR [107] to minimize the re-projection error between 2D keypoints and projected 3D joint positions, as well as the foot contact loss given the predicted foot contact labels. The kinematic motion dataset obtained from this stage is referred to as $M_{kin}$. Examples of the per-frame annotations are visualized in Figure 3.1.

3.3 Physics-Based Motion Correction

Since the estimated kinematic motion dataset ($M_{kin}$) is obtained without explicit modeling of human dynamics, it will contain physically implausible motions such as jitter, foot skating, and ground penetration. To correct these artifacts, we train an imitation policy to control a physically simulated character to track this noisy kinematic motion and output physically corrected motion. The overview of the imitation policy is shown in Figure 3.2.

The approach we use is similar to SimPoE [109]. The task of controlling the character agent in a physically simulated environment to mimic reference motions can be formulated as a Markov decision process (MDP), defined by a tuple $\mathcal{M} = (S, A, T, r, \gamma)$ of states, actions, transition dynamics, a reward function, and a discount factor. We first initialize the state of the simulated character $s_0$.
Figure 3.2: Overview of the imitation policy. Given the current state of the character and a reference motion, the policy specifies the PD targets and residual forces/torques that allow the simulated character to track the reference motion as closely as possible.

to be the same initial state of the reference motion. Starting from $s_0$, the agent iteratively samples actions $a_t \in \mathcal{A}$ according to a policy $\pi(a_t|s_t)$ at each state $s_t \in \mathcal{S}$. The environment then transitions to the next state $s_{t+1}$ according to the transition dynamics $T(s_{t+1}|s_t, a_t)$, and then outputs a scalar reward $r_t$ for that transition. The reward is computed based on how well the simulated motion aligns with the reference motion. The goal of this learning process is to learn an optimal policy $\pi^*$ that maximizes the expected return $J(\pi) = \mathbb{E}_\pi[\sum_t \gamma^t r_t]$. Next, we describe the details of the state, action, and reward function, as well as training strategy of the imitation policy.

### 3.3.1 States

The simulated character model is created based on the SMPL format [53], with body shape parameters estimated using HybrIK [44]. The character consists of 24 rigid bodies and 72 degrees of freedom. We use the following features to represent the character state $s_t = (p_t, \dot{p}_t, q_t, \dot{q}_t, \hat{p}_{t+1}, \hat{q}_{t+1})$:

- $p_t$: joint positions in the character’s root coordinates
- $\dot{p}_t$: joint linear velocities in the character’s root coordinates
- $q_t$: joint rotations in the joints’ local coordinates
- $\dot{q}_t$: joint angular velocities in the joints’ local coordinates
- $\hat{p}_{t+1}$: target (kinematic) joint positions
- $\hat{q}_{t+1}$: target (kinematic) joint rotations

### 3.3.2 Actions

Similar to many prior systems, we use proportional derivative (PD) controllers at each non-root joint to produce torques for actuating the character’s body. The action $a_t$ specifies the target joint
angles $u_t$ for the PD controllers. At each simulation step, the joint torques $\tau_t$ are computed as:

$$\tau_t = k_p \cdot (u_t - q_{nr}^t) - k_d \cdot \dot{q}_{nr}^t,$$

where $k_p$ and $k_d$ denote the parameters of the PD controllers that determine the stiffness and damping of each joint, $q_{nr}^t$ and $\dot{q}_{nr}^t$ are the joint rotations and angular velocities of the non-root joints. To improve tracking performance on highly agile motions, we also allow the policy to apply external residual forces to the root joint \[108\]. Therefore, the actions also include residual forces and torques $\eta_t$ for the root joint, and each action is defined as $a_t = (u_t, \eta_t)$.

### 3.3.3 Rewards

The reward function is designed to encourage the policy to closely track the reference motion while also minimizing energy expenditure. The reward consists of five terms:

$$r_t = \omega_o r_o^t + \omega_v r_v^t + \omega_p r_p^t + \omega_k r_k^t + \omega_e r_e^t.$$  \hspace{1cm} (3.2)

The joint rotation reward $r_o^t$ measures the difference between the local joint rotations of the simulated character $q_j^t$ and the reference motion $\hat{q}_j^t$:

$$r_o^t = \exp \left[ -\alpha_o \sum_j \left( ||q_j^t \ominus \hat{q}_j^t||^2 \right) \right].$$  \hspace{1cm} (3.3)

where $\ominus$ denotes the geodesic distance between two rotations.

The velocity reward $r_v^t$ measures the mismatch between local joint velocities of the simulated motion $\dot{q}_j^t$ and the reference motion $\dot{\hat{q}}_j^t$:

$$r_v^t = \exp \left[ -\alpha_v \sum_j \left( ||\dot{q}_j^t - \dot{\hat{q}}_j^t||^2 \right) \right].$$  \hspace{1cm} (3.4)

The joint position reward $r_p^t$ encourages the 3D world joint positions $x_j^t$ (including the root joint) to match the reference motion $\hat{x}_j^t$:

$$r_p^t = \exp \left[ -\alpha_p \sum_j \left( ||x_j^t - \hat{x}_j^t||^2 \right) \right].$$  \hspace{1cm} (3.5)

The keypoint reward $r_k^t$ encourages the projected 2D joint positions $H_i x_j^t$ to match the detected 2D
keypoints $\mathbf{p}_j^t$:

$$r^k_i = \exp \left[ -\alpha_k \sum_j \left( \| \mathbf{H}_j \mathbf{x}_j^i - \mathbf{p}_j^t \|^2 \right) \right]. \quad (3.6)$$

Finally, the reward $r^e_i$ denotes the power penalty computed as:

$$r^e_i = -\sum_j \left( \| \mathbf{q}^i_j \cdot \tau_j^i \|^2 \right), \quad (3.7)$$

where $\tau_j^i$ is the internal torque applied on the joint $j$. The weight and scale factor for each reward term is manually specified and kept the same in all the experiments.

### 3.3.4 Training

The original imitation policy from Yuan et al. [109] was only trained on the high-quality mocap database AMASS [54]. However, directly applying this policy to track the tennis motion from $M_{\text{kin}}$ will lead to the character falling after a few steps of simulation since the noisy tennis motions differ from the motion examples in AMASS. Therefore, we propose to train the policy in two stages. In the first stage, we train the policy using AMASS to learn to imitate general motions. We then fine-tune the policy using $M_{\text{kin}}$ in the second stage so that it can track the tennis motion more closely without falling. The power penalty $r^e_i$ is only applied during this fine-tuning stage to mitigate the impact of frame-to-frame variances in $M_{\text{kin}}$. Once trained, we can simply run the imitation policy to track each motion sequence from $M_{\text{kin}}$ and export the physically corrected tennis motion dataset, referred as $M_{\text{corr}}$. Note $M_{\text{kin}}$ and $M_{\text{corr}}$ are only collected from the US Open dataset. More details of the rewards, network architecture, and training hyper-parameters can be found in Appendix A.

As mentioned above, the training process yields solutions that rely on residual forces. Reducing this reliance is a challenging problem, but stands to yield more physically accurate motion. To reduce this reliance, we can optionally fine-tune the policy by gradually decreasing the allowed maximum scale of residual forces/torques $\eta_t$ to zero during training. However, the policy without residual forces fail to track fast running and sharp turning motions in our experiments (character falls down). Later in the thesis (Chapter 5), the imitation policy is also used to control the low-level movements of the simulated character to perform new tasks by tracking the target reference motions to form a hierarchical controller. We will further discuss the motion quality and tracking performance trade-off of removing residual forces in Section 5.4.9 when we evaluate the system in an end-to-end manner.
Figure 3.3: Examples of the corrected motion. Top: the simulated character (pink) is able to track the kinematic motion (yellow) closely when controlled by the imitation policy. Bottom: the corrected motion is still inaccurate (wrist and neck motion) when compared to the original video. Note in the leftmost column, the character has a locked wrist and the head is not pointing toward the ball. In the rightmost column, the character stops the racket in the air while the player has the racket stopped over his shoulder.

3.3.5 Results and Discussion

Figure 3.3 (top row) shows sampled frames of the kinematic motion (yellow) and the corrected motion (pink), where the simulated character is capable of tracking the kinematic motion closely when controlled by the imitation policy. Compared to the kinematic motion, the corrected motions are more physically plausible, with clearly more stable foot contacts and less jittering. However, the wrist motion and neck motion still can not be effectively corrected because the estimated motions of these joints are highly inaccurate when compared to the original footage. Note in the leftmost column, the character has a locked wrist and the head is not pointing toward the ball. In the rightmost column, the character stops the racket in the air while the player has the racket stopped over his shoulder.

Therefore, in order to use these motions to build high-fidelity controllers, it requires solutions to hallucinate the inaccurate motion using additional cues, such as the outcome of the physics interaction between racket and ball, or kinematic constraints specific to tennis (e.g. players should keep their eye on the ball). We will describe these solutions in details when introducing the physics-based controller in Chapter 5).

3.4 Manual Annotations

We manually annotate all the tennis specific information where a pre-trained detection model does not exist, including shot cycle boundaries, shot types and outcomes, ball trajectories, and player identities.
CHAPTER 3. ANNOTATING BROADCAST VIDEOS

![Diagram of ball trajectory with annotations](image)

Figure 3.4: Given the space-time location (time, 3D position) of consecutive ball contacts during a rally, we find a ball trajectory that starts at the first space-time contact location, clears the net, bounces in the court, and then closely matches the ending location. We use grid-search over the ball’s initial linear velocity (horizontal and vertical components) and spin velocity to find a matching trajectory.

3.4.1 Shot Cycle Boundaries

To organize the data into the structure of shot cycles, we need to annotate the boundaries for each shot cycle, which are the exact frames when players make ball contacts ($t_i^c$). At the time of this thesis, we manually labeled the ball contact times, which takes approximately the wall-clock time of the original match video.

In order to automate this time-consuming process, we developed an approach for spotting temporally precise, fine-grained events in video (detecting the precise moment in time events occur) [28]. Specifically, we show that ball contact as well as ball bounce in tennis can be spot with Mean Average Precision (mAP) over 96.9% on the test set.

3.4.2 Shot Types and Outcomes

To model the players’ shot selection choices, we identify the type of the shot hit during the shot cycle ($c^i$): groundstrokes, volleys, and serves [78] (differentiating forehands and backhands for groundstrokes and volleys). We also label the outcome of the shot cycle ($o^i$): the player hits the ball in the court without ending the point, hits a winner, hits an error, or does not reach the incoming ball (no contact).

3.5 Ball Trajectory Estimation

Additionally, ball trajectories also contain critical information about players’ shot selection, such as the shot velocity ($v_i^b$), shot spin ($s_i$) and shot placement position ($x_i^b$). However, tennis balls undergo significant motion blur in broadcast footage, as well as occlusion from players, making it difficult to detect the ball trajectory only from image pixels. We take an alternative approach, and
CHAPTER 3. ANNOTATING BROADCAST VIDEOS

Figure 3.5: Estimated trajectories in database clips for all players separated by forehand/backhand shots. The red line indicates the mean of trajectories’ maximum height before bounce. Nadal has the highest trajectories among the three male players.

estimate the ball’s space-time trajectory given the ball contact times and the position and pose of the swinging player during these times. Specifically, we first estimate the 3D position of the ball at the time of contact (shot contact position $x_i^c$) based on the court position and pose of the swinging player. Then we estimate the ball’s trajectory between two consecutive ball contacts in a rally by fitting a pair of ballistic trajectories (with a bounce in between) to the starting and ending contact points (Figure 3.4), using the ball flight model described in Section 2.2. Given the time and 3D positions of two consecutive ball contacts in a rally, we perform a grid search over the components of the ball’s launch velocity (horizontal and vertical components of linear velocity, as well as the launch spin) to yield a trajectory which starts at the first contact point, clears the net, bounces in the court, and then closely matches the time and location of the second contact point. This trajectory determines the ball’s position and velocity at all times in between the two contacts, including the shot velocity as well as the time and location of ball’s bounce. In cases where the second ball contact is a volley, we adjust the search algorithm to find a trajectory that travels directly between these contact times/locations without a bounce. Note that the accuracy of the ball trajectory estimation can be further improved if additional constraints are provided, such as partial trajectory detection or annotations of ball bounce times/locations. In Figure 3.5, we plot all the estimated ball trajectories for the four players from the Wimbledon dataset. Overall, it suggests that the estimated ball trajectories are reasonable and indicate similar trends as the real world players. For example, Nadal has the highest trajectories among the three male players, which is caused by his aggressive use of spins.
Chapter 4

Tennis Control

Given the pipeline of synthesizing a tennis point in Section 2.3, this chapter focuses on the tennis control problem and begins with a more detailed problem definition. Subsequently, we conduct a brief review of the existing literature on human motion controllers, with a particular focus on data-driven approaches and discuss the underlying motivations behind the controllers we proposed for addressing the tennis control problem. Finally, we present a simple kinematic, non-parametric controller inspired by Motion Matching [11]. We demonstrate that by leveraging domain knowledge of tennis, specifically the shot-cycle state machine for organizing the motion database and determining the motion transitions, we can achieve high-quality results by solving the tennis control problem via k-NN search, and producing player animation by stitching together shot-cycle motion clips. We will discuss the limitations of this simple controller which motivates a more sophisticated physics-based controller in Chapter 5. Later in this thesis, we will also demonstrate that the simple controller can be extended to a video sprite based animation system to incorporate appearance information in Chapter 8.

4.1 Problem

Given the point synthesis pipeline described in Section 2.3, we divide the tennis rallies into shot-cycles and formulate the tennis control problem within each cycle. Here we provide a more detailed definition of the tennis control problem. At the beginning of each cycle, which marks the moment when the opponent has just hit the ball, the state inputs include the player’s current pose \( \mathbf{p}_0 \), court position \( \mathbf{x}_\text{p0} \), and root velocity \( \mathbf{v}_\text{p0} \), along with the incoming ball’s trajectory \( \mathbf{b}_\text{ball}(t) \). We simplify the problem by assuming the entire ball trajectory is known at this time given the fact that professional tennis players can estimate the ball trajectory based on the ball’s launch state. In our system, this estimation can be achieved by running the ball flight simulation forward in time.

In addition to the state inputs, the problem also contains a set of motion goals \( G^* = (c^*, v^*_c, \mathbf{x}_b^*, \mathbf{x}_r^*) \),
such as shot selection goals, including shot type ($c^*$), shot velocity ($v^*_b$) and shot placement position ($x^*_b$), as well as a recovery position goal ($x^*_r$). All the problem inputs are illustrated in Figure 4.1.

The desired output of this control problem is a sequence of motion which aims to:

- Start at the player’s current position $x_{p0}$, and exhibit good visual continuity with the player’s current pose $p_0$ and velocity $v_{p0}$.
- Make the player swing at a time and court position that places the racket in the incoming path of the ball.
- Produce a shot that matches type $c^*$, launches with spin $s^*$ and bounces near $x^*_b$.
- Move the player to $x^*_r$ after ball contact.

To control two players in a full tennis point, we can alternatively solve this control problem for each player repeatedly until the point ends. Two special cases of the control problem arise at the beginning and end of the point. At the beginning of the point, the server encounters a ball trajectory thrown into the air instead of incoming from the other side. In our system a point ends when one of two events occur: a player makes an error by hitting the ball outside the court, or a player hits a shot that lands in the court but cannot be reached by the other player. When the point ends, there is no recovery position goal and the output motion should transition into motion depicting player stopping play.

## 4.2 Related Work

We briefly review the literature of human motion controllers focusing on data-driven approaches and discuss the motivations of the controllers we proposed for the tennis control problem. We organize the prior work as kinematic control and physics-based control.
4.2.1 Kinematic Control

These approaches aim to generate continuous human motion following specified control without the explicit use of a physics simulator. Early approaches are non-parametric which directly use the motion data to construct generated motion. More recently, parametric methods using deep neural networks (DNN) are capable of synthesizing motion in a pose by pose manner and the original motion data can be discarded after training.

We further organize the kinematic control methods into direct prediction vs. model-then-control: whether the example motion data is used along with the control task to form a one-stage approach, or the motion data is first modeled and then the control can be solved via search.

**Direct prediction.** In this category, integration of example motion data into the task-specific solutions is a common practice. One such approach is Motion Matching, [11], where the best transition into a new clip from the database is selected using a k-NN query at each frame. The locomotion task is embedded into the feature vector, incorporating both the past root trajectory and the desired goal such as future trajectory. This technique ensures the continuation of ongoing movement while fulfilling the task’s objectives.

Similarly, motion clips can also be segmented at coarse grained action boundaries to reduce the search space, as shown in a prior tennis controller [36]. In the next section 4.3, we introduce a simple non-parametric tennis controller similar to Lai et al., but utilize the shot-cycle state machine to determine the clip transitions and extend the k-NN query to involve behavior constraints to solve the tennis control problem.

Supervised learning, facilitated by Deep Neural Networks (DNNs), offers an end-to-end training solution for the control problems. Autoregressive DNN models have been employed for various scenarios, such as human variable-terrain locomotion [26], quadruped variable-terrain locomotion [110], and environment-aware human locomotion involving interactions like carrying an object or sitting on a chair [79]. Recurrent Neural Networks (RNNs) have also been utilized, with extensive data augmentation and objective annotation, for control problems in locomotion, basketball, and tennis [40]. Learned Motion Matching [25] reimplements Motion Matching with DNN to replace the k-NN query and substantially decreases the memory footprint. It is notable that these approaches primarily focus on locomotion control tasks, such as trajectory following, without strict spatial-temporal constraints. Therefore, they are well-suited for supervised learning since dense supervision can be derived from the motion database. In the case of longer-term objectives, such as carrying an object at a specific location, extensive data augmentation with hand-crafted rules are essential for achieving success [40, 79].

**Model-then-control.** Different from direct prediction, an alternative approach is to first develop a model representing the space of possible motions, and then solve the control task by searching for
an optimal path in the motion model to achieve the desired task goal. Motion Graph [35] is an early approach of explicitly building a graph connecting possible transitions between arbitrary frames of database motion clips. It can also be useful to segment the motion into short clips which covers full walk cycles so that transitions between any clips form valid animations without foot skating [85]. Kinematic controllers can then be developed by traversing the graph to create novel motion that best satisfies the task goal, such as boxing [39], locomotion and grasping [52].

More recently, there has been a growing focus on generic parametric motion models that are trained to enable sampling from a distribution of future states. These models can then serve as the foundation for constructing motion planners or control policies to achieve multiple desired tasks. Min and Chai [57] employ graph traversal and probabilistic sampling techniques to synthesize motions from contact-aware Gaussian mixture motion primitives. Holden et al. [27] propose to first learn a latent motion manifold using Convolutional Autoencoders, then map the locomotion control signals to the latent variables via supervised learning. The advancement of generative models has played a significant role in enhancing the learning of motion transitions that capture the inherent variability in human movements. Various models have been leveraged, including Recurrent Convolutional Variational Autoencoder [21], Generative Adversarial Network (GAN) [89], Conditional Variational Autoencoder (cVAE) [45], Normalizing Flows [23] and Diffusion models [83]. In particular, our tennis controller introduced in Chapter 5 adopts the cVAE model similar to Ling et al. [45]. This choice allows us to efficiently model the tennis motion space from large-scale motion data, and the learned distribution can be easily utilized by a motion planner using deep reinforcement learning (DRL).

4.2.2 Physics-Based Control

In contrast to kinematic motion models that replicate motions resembling those in the dataset, physics-based motion control offers the advantage of producing realistic dynamic responses to external forces and environmental changes. By incorporating principles of physics and utilizing a physics simulator, the complexity of hand-crafted constraints in kinematic control can be significantly reduced. For instance, in kinematic control, additional post-processing steps such as foot-skate cleanup or the specification of thresholds for fake object interactions are often required to achieve a semblance of realism. In contrast, a physics-based approach inherently accounts for these interactions, eliminating the need for such ad-hoc constraints.

Early approaches have leveraged optimization techniques with hand-crafted control structures, such as finite state machines, to create controllers for a large variety of complex behaviors [14, 15, 24, 43, 48, 58, 69, 81, 102, 103]. More recently, DRL has been used to drastically expand the corpus of skills that can be reproduced by simulated characters, including locomotion [64, 74, 95, 106], dressing [12], object manipulation [56], and recovery behaviors [82].

The difficulty of designing reward functions that lead to lifelike motions has motivated the use of motion imitation techniques, where naturalistic motion can be learned by imitating reference motion
The access to specialized motion capture data have demonstrated impressive results for training controllers for more complex athletic skills, such as basketball dribbling, soccer dribbling and shooting, figure skating, multi-agent soccer, boxing and fencing, and soccer juggling.

Learning skills from videos. Mocap has been the most commonly used source of motion data for physics-based character animation. While mocap can provide high-quality motion data, these systems typically require heavy instrumentation of both the environment and actors, which can be difficult to apply to large-scale outdoor sports. Video clips can provide a much more abundant and accessible source of motion data. Vondrak et al. [86] represents one of the first physics-based character animation systems for video imitation, using hand-designed FSM controllers and an incremental optimization approach with a 2D-silhouette matching objective. Peng et al. [66] combine 2D/3D pose estimators and deep reinforcement learning to train controllers to imitate a diverse corpus of skills from short video clips. Yu et al. [105] extend this approach to imitate longer video sequences with dynamic camera movements and irregular environments. Yuan et al. [108] propose residual force control to compensate for the dynamics mismatch between simulated characters and real humans, which enables tracking of agile motions such as ballet dance. Most recently, SimPoE [109] trained general motion tracking controllers using large-scale mocap dataset (e.g., AMASS [54]), which enables zero-shot imitation of skills from monocular videos in real-time. However, the prior systems [66, 105] are limited to reproducing the same skill from short video clips and require learning an individual policy for each skill, which restrict the potential use case of imitation from a large-scale video database like the one we collected from broadcast tennis videos.

Hierarchical control. Following the philosophy of model-then-control, low-level imitation controller can be expanded from a single example to a diverse motion dataset, by combining with kinematic controllers to form a hierarchical framework used to direct a character to accomplish more complex tasks which may require composition of multiple skills in furtherance of a desired task objective. Bergamin et al. [2] use motion-matching to generate plausible motion trajectories for guiding a physically simulated character to perform a given task, while Park et al. [61] train an autoregressive motion prediction model to generate plausible human motions. More recently, generative motion models are adopted to form the motion model. These include employing Motion Graphs with mixture-of-expert networks [92], using cVAEs [94, 100], or using GANs [65, 67, 98].

Our proposed video imitation system is also built upon the hierarchical control approach, where a motion embedding is first constructed using cVAE from the tennis motion dataset collected from broadcast videos. The learned low-dimensional motion embedding reduces the search space of DRL and serves as a basis for synthesizing reference motion that achieves the given task. These reference motions are finally imitated by the low-level imitation controller, where the use of the physics-based controller significantly simplifies the design of the reward function in DRL given the outcome.
Algorithm 1 Point synthesis algorithm of the simple controller

1: Let $P_{cur}$ be the serving player, $P_{opp}$ be the returning player
2: Perform clip search for $P_{cur}$ (serve clip only)
3: Swap $P_{cur}$, $P_{opp}$
4: while True do
5: (Start of the shot cycle for $P_{cur}$, $P_{opp}$ has just made ball contact)
6: Determine if $P_{cur}$’s shot will end point (ERROR or WINNER)
7: Perform clip search for $P_{cur}$
8: Adjust recovery phase of $P_{opp}$
9: BREAK if point ends
10: Swap $P_{cur}$, $P_{opp}$
11: end while
12: Perform clip search for $P_{opp}$ (point-ending clip only)

of the physics interaction (racket-ball contact), without the need for hand-crafted constraints or thresholds.

4.3 A Simple Kinematic Controller

In this section, we introduce a simple kinematic, non-parametric controller to address the aforementioned tennis control problem. The kinematic controller adheres to the paradigm of direct prediction, which is inspired by Motion Matching [11]. We demonstrate that by leveraging domain knowledge of tennis, specifically the shot-cycle state machine for organizing the motion database and determining the motion transitions, we can achieve high-quality results by solving the tennis control problem via k-NN search, and producing player animation by stitching together shot-cycle motion clips. We will discuss the limitations of this simple controller which motivates a more sophisticated physics-based controller in Chapter 5. Later in this thesis, we will also demonstrate that the simple controller can be extended to a video sprite based animation system to incorporate appearance information in Chapter 8.

4.3.1 Overview

Given the tennis motion dataset collected in Chapter 3, the motion database can be organized as a collection of shot-cycle motion clips. Each clip is segmented at the shot cycle boundaries, which are the times when the opponent makes ball contact. Given the clip database, the control problem introduced in Section 4.1 can be solved by searching for a clip that best meets all the constraints at the beginning of each shot cycle. The pseudocode of the full point synthesis algorithm employing clip search is shown in Algorithm 1.

The kinematic controller basically implements the Motion Matching [11] algorithm, but with transitions only allowed at the shot cycle boundaries. This yields several benefits. First, placing
transitions at the end of the shot cycle as the player nears the ready position makes discontinuities harder to perceive, since this is the moment where tennis players make abrupt changes in motion as they identify and react to the direction of an incoming shot. It is also the moment when the opposing player is hitting the ball, so the viewer’s gaze is likely to be directed at the ball rather than at the player undergoing a clip transition. Second, performing clip transitions at shot-cycle granularity facilitates interactive performance, since search is performed infrequently and only requires identification of single matching database clip (not a full motion graph search). Finally, making all control decisions at the start of the shot cycle aligns with the moment when real-life tennis players make key decisions about play. This facilitates modeling the behavior by allowing clip selection to consider goals for all phases of the shot cycle: shot preparation (reaction), swing, and recovery to identify a continuous segment of database clip that globally meet these goals (discussed in Chapter 7 with more details).

Note that the controller described in the rest of this section is presented with a 2.5D motion representation, where the player’s root positions and ball positions are in 3D court space while the player’s poses are represented using 2D keypoints in screen space. However, extending the controller to 3D skeletal motion is straightforward since none of the algorithm component is specific to 2D or 3D.

Since it is unlikely that any database clip will precisely meet all conditions, we first describe clip manipulations that improve a clip’s match with specified constraints (Section 4.3.2, Section 4.3.3), then describe the proposed cost model (Section 4.3.4). We also cover details of how clip search is used to determine when points end (Section 4.3.5).

4.3.2 Making Ball Contact

During the reaction phase of the shot cycle, the goal is to produce an animation that moves the player from $x_{p0}$ to a position where their swing makes contact with the incoming ball. Rather than specify a specific ball contact point on the incoming trajectory, we allow the player to hit the ball at any feasible point on its trajectory. For each clip $i$ in the database, we use the length of the reaction phase to determine the incoming ball’s position at the clip’s ball contact time $x_{ball}(t^i_c)$. After translating the player in the clip in court space to start the shot cycle at $x_{p0}$, the player’s shot contact position (the position of the racket head) resulting from the clip is given by:

$$x_c = x_{p0} + (x^i_c - x^i_p(0)).$$  \hspace{1cm} (4.1)

For the rest of the paper, we use superscript $i$ notation to refer to positions/times from source database clips. Variables without a superscript denote positions/times in the synthesized point.
CHAPTER 4. TENNIS CONTROL

27

Incoming ball
Player trajectory (reaction phase) from database clip
Player trajectory (recovery phase) from database clip
Corrected player trajectory (reaction phase)
Corrected player trajectory (recovery phase)
Racket head (ball contact position)

Figure 4.2: Translational correction in clip search. Using motion directly from a database clip would place the player’s racket at $x_c$ at ball contact time, and the player at $x_r$ at the end of the shot cycle. We compute the error in these positions $e_c$ (the distance from $x_c$ to the ball) and $e_r$ (distance to the target recovery position $x^*_r$), and apply a translational correction to the root position of the player so that the racket meets the incoming ball, and that player reaches the target recovery position at the end of the shot cycle.

Translational Correction  Directly using the player’s motion from the database clip is unlikely to make the player’s swing contact the incoming ball. To make the player appear to hit the ball at the clip’s ball contact time ($t^i_c$), we add a court-space translational correction to move the player so that the racket head is located at the same location as the ball at the time of contact (Figure 4.2). We compute the position error in the racket head:

$$e_c = x_{ball}(t^i_c) - x_c,$$

and modify the player’s motion in the clip with a court space translational correction that places the player at the correct spot on the court to hit the ball. This correction is increasingly applied throughout the reaction phase by linearly interpolating the uncorrected and corrected motions. This yields a corrected player court space position for the reaction phase:

$$x_p(t) = x_{p0} + (x_{p}^r(t) - x_{p}^r(0)) + w(t/t^i_c) e_{c2d},$$

where $w(x)$ is an ease-in, ease-out function used to generate interpolation weights, and $e_{c2d} = [e_c.x, e_c.y, 0]^T$. Since translational correction does not move the player off the ground plane, it does not eliminate the $z$ component (height) of contact point error.
A large translational correction can result in objectionable foot skate, so clip search aims to minimize $e_{c2d}$ (Section 4.3.4). If there is no clip in the database with a sufficiently small translational correction, the player cannot reach the ball, signaling the end of the point (Section 4.3.5).

### 4.3.3 Meeting Recovery Constraints

For the recovery phase of the shot cycle, we compute a translational correction to move the player’s position at the end of the recovery phase $x_p(t_r^i)$ to the target recovery position $x_*^r$. This correction is computed similarly to the correction used to ensure ball contact (Figure 4.2):

\[
x_r = x_p(t_r^i) + (x_p(t_r^i) - x_p(t_i^c))
\]

\[
e_r = x_*^r - x_r.
\]

Note from the pseudocode in Algorithm 1 that the actual length of the recovery phase of the shot cycle initiated in this iteration is not known when clip search occurs (line 7). The length is not determined until the next loop iteration, when clip search determines when the opponent will hit the return shot. Therefore, when evaluating clip cost during search, we assume the length of the new shot cycle’s recovery phase is the same as that of the database clip $(t_r^i - t_i^c)$ when computing $e_r$. When the opponent’s future ball contact time is determined (in the next loop iteration), we compute the actual $e_r$ for the chosen clip by replacing $t_i^c$ with this time in Equation 4.4 (Algorithm 1, line 8). The actual $e_r$ is used to compute the translational correction used for recovery phase sprite rendering.

To avoid excessive foot skate during recovery, we constrain $e_r$ to a visually acceptable range (see $C_{recover}$ in Section 4.3.4). Therefore, unlike the reaction phase, where translation is used to bring the player’s racket exactly into the trajectory of the incoming ball to continue the point, the recovery translation may not always move the player exactly to the target recovery position $x_*^r$. We judged that failing to precisely meet the recovery goal was acceptable in times when doing so would create notable visual artifacts.

### 4.3.4 Clip Cost

We compute the cost of each database clip as a weighted combination of terms that assess the continuity of motion between clips, the amount of manipulation needed to ensure ball contact and meet recovery position goals, as well as how well a clip matches the shot velocity and placement components of shot selection goals.

**Pose match continuity.** ($C_{pose}$) We compute the average screen-space $L_2$ distance of joint positions between the current pose (the last frame of the previous shot cycle) $p_0$ and the starting pose of the database clip $p^i(0)$. Since the size of a player’s screen projection depends on their court location
when using 2.5D motion representation, we can scale \( p^i(0) \) by a factor \( \sigma \) computed from the court positions of the two poses to normalize the screen-space size of poses (details about \( \sigma \) in Chapter 8).

\[
C_{\text{pose}} = \| p_0 - \sigma p^i(0) \|.
\]

**Player root velocity continuity.** (\( C_{\text{velo}} \)). We compute the difference between the player’s current velocity \( v_{p0} \) and the initial velocity of the database clip \( v^i_p(0) \). \( (v^i_p(t) \) is computed by differencing \( x^i_p(t) \) in neighboring frames.)

\[
C_{\text{velo}} = \| v_{p0} - v^i_p(0) \|.
\]

**Ball contact height matching.** (\( C_{\text{contact}} \)) We use this term to force the player’s racket height at contact to be close to the ball’s height. (Recall that the correction \( e_{c2d} \) only corrects error in the racket’s XY distance from the ball.)

\[
C_{\text{contact}} = \| x_c.z - x_{\text{ball}}(t^i_c).z \|.
\]

**Reaction phase correction.** (\( C_{\text{react}} \)). This term assigns cost to the translational correction applied to make the player contact the ball. We observe that high velocity clips can undergo greater correction without noticeable artifacts, so we seek to keep velocity change under a specified ratio. Also, translations that significantly change the overall direction of a player’s motion in the source clip are easily noticeable, so we penalize translations that create these deviations. This results in the following two cost terms:

\[
C_{\text{react,velo}} = \max \left( \frac{\| \Delta x_{\text{cor}} \|}{\| \Delta x_{db} \|}, \frac{\| \Delta x_{db} \|}{\| \Delta x_{\text{cor}} \|} \right)
\]

\[
C_{\text{react,dir}} = 1 - \cos(\Delta x_{db}, \Delta x_{\text{cor}}),
\]

where \( \Delta x_{db} = x^i_p(t^i_c) - x^i_p(0) \) and \( \Delta x_{\text{cor}} = x_p(t^i_c) - x_{p0} \) give the player’s uncorrected and translation corrected movement during the reaction phase of the database clip.

**Recovery phase correction.** (\( C_{\text{recover}} \)). The term measures the amount of correction needed to achieve the target player recovery position, and it is computed in the same way as \( C_{\text{react}} \).

**Shot match.** (\( C_{\text{shot}} \)) This term ensures that the chosen clip matches the target shot type \( c^* \) and also measures how closely the shot velocity \( v^i_b \) (derived from the estimated ball trajectory) and placement position \( x^i_b \) of the shot in the database clip match the behavior goals \( v^*_b \) and \( x^*_b \) (after
accounting for player translation).

\[
C_{\text{shot, type}} = \begin{cases} 
0 & \text{if } c^i = c^s \\
\infty & \text{otherwise} 
\end{cases}
\]

\[
C_{\text{shot, velo}} = \|v^i_b - v^s_b\|
\]

\[
C_{\text{shot, place}} = \|x^i_b - (x^i_b + (x_p(t^i_c) - x_p(t^s_c)))\|
\]

**Total Cost** Since these cost terms are measured in different units (e.g. \(C_{\text{pose}}\) in pixel distance and \(C_{\text{contact}}\) in meter). To make them roughly comparable, for each cost term we determine a threshold for visually unacceptable results and use the threshold to normalize the cost term following the equation below:

\[
C' = \begin{cases} 
C/C^t & \text{if } C < C^t \\
\infty & \text{otherwise} 
\end{cases}
\]

where \(C\) and \(C'\) denote the cost before and after normalization, \(C^t\) denotes the threshold. We list the thresholds and weights in Table 4.1. Once all the cost terms are normalized, we linearly combine them using weights that balance trade-offs between different constraints. We give the highest weight to \(C_{\text{recover, dir}}\) since recovering in the wrong direction yields extremely implausible player behavior. We also prioritize the weights for \(C_{\text{react, velo}}\) and \(C_{\text{react, dir}}\) to reduce foot skate during the reaction phase. We give lower weight to \(C_{\text{contact}}\) since the fast motion of a swing (the racket is motion blurred) makes it challenging to notice mismatches between the position of the ball and racket head at contact, and to \(C_{\text{shot, place}}\) since inconsistencies between swing motion and the direction the simulated ball travels typically require tennis expertise to perceive. The final weights are determined experimentally to produce good results.

### 4.3.5 Point Ending

In our system a point ends when one of two events occur: a player makes an error by hitting the ball outside the court, or a player hits a shot that lands in the court but cannot be reached by the other player. When errors occur can be determined randomly given a user specified probability, or determined using the shot selection behavior model described in Chapter 7. We use clip search and
CHAPTER 4. TENNIS CONTROL

Figure 4.3: Results of the simple controller. (a) The character starts with a ready pose at the beginning of the shot cycle (b) The character hits the ball along the incoming ball trajectory with target shot type (c) The outgoing ball matches the target shot spin (d) The outgoing ball bounces at the target shot placement position (e) The character recovers to the target recovery position at the end of the shot cycle.

the amount of required recovery phase translation correction to determine when a player cannot reach an incoming ball. If no clip in the database has sufficiently low $C_{\text{react}}$, then the player cannot reach the ball and the system determines they lose the point.

The point synthesis algorithm determines that the simulated point will end in the current shot cycle before performing clip search (Algorithm 1, line 6). This allows database clips to be excluded from clip search if they do not match the desired shot cycle outcome (as determined by the clip’s shot outcome $o_i$). For example, when the current shot does not cause the point to end, clip search is limited to database clips that are not point ending shot cycles. These clips depict the player rapidly returning to a ready position, poised to react to the next shot. If the player cannot reach the ball, search is limited to only database clips with the same outcome, resulting in output where the player concedes the point without making ball contact. Finally, since there are only a small number of clips in the database depicting point ending shots (at most one clip per point), if the player will make an error or hit a winner, we search over all clips where ball contact is made (both point ending and not point ending), choosing one that minimizes the total cost. Although not shown in Algorithm 1’s pseudocode, if the best clip does not depict a point ending shot cycle, after ball contact we insert an additional clip transition to the recovery phase of a point ending clip to depict the player stopping play at the end of the final shot cycle.
Sensitivity to clip database size. To understand how clip search cost decreases with increasing amounts of source video content, we measure the average value of components of the clip search cost metric (Section 4.3.4) as the database size is increased (Figure 4.4). (We compute the average clip search cost over 400 simulated rallies for each player.) Federer’s results converge most quickly, suggesting that his iconic smooth, “always well positioned” playing style yields a narrower distribution of real-life configurations. Similarly, Nadal’s costs converge less quickly, and overall remain higher than Federer and Djokovic’s, indicating greater diversity in his on-court play.

Database Clip Usage. We also evaluate how frequently database clips are used during our novel point synthesis. For this evaluation, we randomly generate 1500 points between the three male players. Fig. 4.5 plots a histogram of the number of times each clip was used in these simulations. 89.1% of the database clips are used at least once. The most frequently selected clips are cross-court...
CHAPTER 4. TENNIS CONTROL

Figure 4.5: The statistics of how frequently database clips are used during the synthesis of 1500 randomly generated points. Almost 90% of the database clips are used at least once.

ground stroke rallies, which is the most common pattern in tennis. Never chosen clips are in general depicting atypical behaviors, especially net play, for which corresponding game situations are less likely to occur in our synthesis.

Limitations. The simple k-NN controller has two major limitations. First, discontinuities may occur at the clip transitions and the translation correction may introduce sliding artifacts. Although the cost metrics ($C_{\text{pose}}$ and $C_{\text{react}}$) attempt to minimize these artifacts, with only a finite amount of examples in the database, and manipulation of the source motion only at the granularity of shot-cycle clips, these artifacts will always occur to some degree. It motivates us to develop a parametric controller which models and generates motion at the granularity of frames. Additionally, the kinematic controller requires hand-crafted constraints ($C_{\text{shot}}$) to ensure visually plausible interaction at racket-ball contact, but may still lead to mismatch between the swing motion and the outgoing ball trajectory. This motivates a physics-based controller to simulate racket-ball contact during the interaction, which can significantly improve the realism of the interaction and alleviate the need of hand-crafted constraints.
Chapter 5

Learning Physics-Based Tennis Control

In this chapter, we propose a hierarchical physics-based controller building upon recent ideas in data-driven and physics-based character control. Our solution leverages motions produced by physics-based imitation of example videos to learn a rich kinematic motion embedding for tennis actions, then trains a high-level motion controller that steers the character in the latent motion space to produce kinematic motions that achieve higher-level task objectives (e.g., hitting an incoming tennis ball). Finally, we use the imitation controller (the policy used for motion correction in Section 3.3) to control the low-level movement of the simulated character by imitating the target kinematic motion. To address motion quality issues caused by perception errors that persist in the learned motion embedding (e.g., blurred or occluded wrist motion, inaccurate neck rotations, as discussed in Section 3.3.5), our pipeline overrides erroneous reference motion with physics-based corrections driven by high-level task rewards or by using simple kinematic constraints specific to tennis (e.g., players should keep their eye on the ball). The hierarchical controller utilizes residual force control [108] in the low-level imitation policy to increase overall task performance, but can also function without residual forces while incurring only modest task performance reductions. We demonstrate controllers for physically-simulated tennis players that can hit the ball to target positions on the court with high accuracy and can successfully conduct competitive rally play that includes a range of shot types and spins, as shown in Figure 5.1.

5.1 Overview

Considering the limitations of the simple kinematic controller discussed in Section 4.3.6, we are motivated to develop a more sophisticated controller which has the following two properties. First,
to account for the sparsity of ground-truth control problems (i.e. incoming ball trajectory and the corresponding reacting motion from the video), reinforcement learning (RL) is favored than supervised learning approach. To further reduce the search space of RL and facilitate training, the controller should adopt the paradigm of model-then-control which first constrains the space of possible actions to a set of human tennis motions by leveraging the large-scale unstructured motion data to build a motion model. The RL-based controller can then be trained by practicing with a large number of randomly generated control problems (i.e. randomly generated incoming balls). Second, to simplify the reward design for the RL-based controller, it is helpful to develop the controller in a physics-based environment to harness the simulated physical interactions and the observations of its outcome (e.g., outgoing ball trajectory). The same reward can also be utilized to hallucinate the inaccurate wrist motion from the motion model.

An overview of the proposed hierarchical controller is shown in Figure 5.2. The training process takes as input the reconstructed motion datasets of different players, and outputs controllers for physically simulated characters that hit consecutive incoming tennis balls using a diverse set of tennis skills. The controllers can be used to produce 3D character animation depicting two simulated characters playing tennis rallies.

Given the physically corrected motion dataset (Section 3.3), we fit a conditional VAE to the corrected motion dataset to learn a low-dimensional motion embedding that produces human-like tennis motions (Section 5.2). A high-level motion planning policy is then trained to generate target kinematic motion by combining body motion from the motion embedding with predicted corrections for the character’s wrist motion (Section 5.3). Finally, the target motion is imitated by the low-level imitation policy, which is the one we used for motion correction (Section 3.3), to control a physically simulated character to perform the desired task.
5.2 Motion Embedding

Given the physically corrected motion dataset $M_{corr}$, we build a kinematic motion embedding to serve as the action space for subsequent high-level planning of long-term tennis motions. This generative model is instantiated using conditional VAE, which learns a low-dimensional latent space using $M_{corr}$.

Our motion embedding model is based on the motion VAE (MVAE) model [45]. An overview of the motion embedding model is shown in Figure 5.3. Given the character’s current pose and a latent variable $z$ representing possible transitions from the current pose, MVAE reconstructs the pose in the next time step while shaping $z$ into a normal distribution. At run-time, the encoder is discarded and the decoder takes the input of the current pose and a latent $z$ to produce the next pose. The predicted next pose can be used as the input in the next step to generate a sequence of poses autoregressively. Our system adapts the original MVAE in two ways: (1) to prevent global drift, we condition on pose features in global court coordinates and (2) in addition to the pose, our model also predicts the phase of the character’s motion (tennis motion is cyclic from shot to shot). Phase information simplifies the design of the reward functions of the high-level policy.

5.2.1 Pose Representation

The pose in each frame is represented using the following features:

- $q_t^r$: root orientation in the global court coordinates
- $r_t$: root position in the global court coordinates
- $\dot{r}_t$: root linear velocity in the global court coordinates
- $p_t$: joint positions relative to the root in global court coordinates

Figure 5.2: Overview of the hierarchical controller. We fit a conditional VAE to the corrected motion to learn a motion embedding that produces diverse and human-like tennis motions. A high-level motion planning policy is then trained to generate target kinematic motion by predicting VAE latent codes and joint corrections for wrist motion. Finally, the target motion is imitated by the low-level imitation policy to control a physically simulated character to perform the desired task.
Figure 5.3: Overview of the motion embedding model. At training time, given the character’s current pose and a latent variable $z$, the model reconstructs the pose in the next time step and outputs the motion phase while shaping $z$ into a normal distribution. At test time, the encoder is discarded and the decoder takes the input of the current pose and a latent $z$ to produce the next pose and motion phase.

- $\dot{p}_t$: linear joint velocities in the global court coordinates
- $q_L$: joint rotations in the local joint coordinates

Representing positions/orientations in the global court coordinates provide a strong prior to the tennis motion. For example, backhand motions are more likely to be performed on the left side of the court for a right-handed player and players should be facing toward the net after each shot. In practice, we find that conditioning motion generation on global positions/orientations yields motion that more consistently recovers back toward the center of the court and has the character face the net as they ready themselves for the next shot.

5.2.2 Motion Phase

Tennis players undergo cyclic motion from shot-to-shot during a point, as introduced in Section 2.1. Knowing the current phase of this motion simplifies the design of reward functions for the high-level policy which is responsible for long-term motion planning. For example, a reward can be designed to minimize the distance between the racket and the ball at ball contact time. Therefore, we adapt MVAE to also predict the motion phase for the output pose. Specifically, we represent the motion phase at each frame with a cyclic phase variable $\theta$ in $[0, 2\pi]$ based on the shot-cycle state machine. $\theta = \pi$ denotes when the player makes ball contact and $\theta = 0$ or $\theta = 2\pi$ denotes...
when the player recovers (the opponent makes ball contact). The phase for the rest of the frames is linearly interpolated between the neighboring two anchors. To avoid a discontinuity at $\theta = 2\pi$, we encode the motion phase with $\sin \theta$ and $\cos \theta$. Due to the repetitive structure of tennis motion, the motion phase can be reliably learned in a semi-supervised setting by providing phase information for a sparse sampling of shots (we annotated only 20% of all shots). We find that the model’s predicted phase is always close to $\pi$ when the corresponding swing motion nears the point where the ball should be contacted.

5.2.3 Training

We follow the network design and general training setup of MVAE [45] and incorporate a number of strategies that are crucial to successfully training a model on the reconstructed tennis motions. Since the input motion data from $M_{corr}$ is still noisier than mocap data, MVAE tends to be more susceptible to error accumulation when generating longer sequences at run-time. To improve the stability of the autoregressive predictions, we follow Ling et al. [45] and adopt scheduled sampling [1], which starts with using ground-truth pose as input and gradually shifts to using predicted pose as input. In our experiments, we find that the selection of the coefficient $\beta$ for the KL divergence loss is critical for learning a good motion embedding for use by the high-level policy. When $\beta$ is too large, the decoder will ignore the latent variable $z$ and only playback the original motion data. When $\beta$ is too small, MVAE may overgeneralize and produce implausible motions with clear artifacts such as foot skate. Empirically, we find that $\beta = 0.5$ effectively balances the flexibility and motion quality of the learned motion embedding. More details of the training process are provided in Appendix A.

5.3 High-Level Motion Planning Policy

Given a motion embedding capable of producing a diverse set of tennis motions, we train a high-level motion planning policy that synthesizes novel motions that enable a character to perform tasks such as hitting an incoming tennis ball to specific target locations. The high-level policy selects latents from the motion embedding to generate kinematic motion trajectories that resemble human behaviors [45]. The resulting kinematic motions are then used as target reference trajectories to drive a physically simulated character using the low-level imitation policy trained in Section 3.3.

However, directly applying the aforementioned approach will fail to produce characters that successfully hit the ball back into the court with a high degree of success. The problem is that even with physics correction, occlusions and motion blur result in $M_{corr}$ containing notable errors in the character’s estimated wrist motion (and corresponding racket motion). Even small errors in swing motion can prevent the high-level policy from finding motion solutions that accurately hit the ball. To overcome inaccuracies in the reconstructed motion data, we propose a hybrid control approach where the full-body motion is controlled by the reference trajectories generated by the MVAE, while
CHAPTER 5. LEARNING PHYSICS-BASED TENNIS CONTROL

Figure 5.4: Overview of the high-level planning policy. The high-level motion planning policy is trained to generate target kinematic motion by combining body motion from the motion embedding with predicted corrections for the character’s wrist motion. The target motion is then imitated by the low-level imitation policy to control a physically simulated character to perform the desired task.

the wrist motion is directly controlled by the high-level policy. We optimize the high-level policy using a curriculum curated for tennis play.

In addition to swing motion errors, the reconstructed tennis motions feature additional errors such as the character’s eyes not tracking the ball and the character’s non-dominant hand not gripping the racket during two-handed swings. Since addressing these artifacts would require additional reward engineering, we propose alternative solutions using simple kinematic constraints specific to tennis.

5.3.1 Policy Representation

The problem of jointly optimizing the predicted MVAE latent codes and predicted joint corrections can be formulated as an MDP and solved with reinforcement learning. We now describe the details of the state and action representations used for the high-level policy.

States. The state consists of a set of features that describes the state of the character and the incoming ball, as well as control targets specified by the system. The character state shares the same pose representation used for the MVAE from Section 5.2, however all features are now computed from the simulated character instead of the kinematic motion. The ball state is represented using the ball’s position in the next ten frames (including the current position), which provides the policy with a forecast of the ball’s future trajectory. The future trajectory of the ball is estimated given the ball’s launch velocity, spin, and height (more details can be found in Appendix A). Control targets consist of the desired placement of the character’s next shot (the position where the ball should bounce on the other side of the court) and a binary variable indicating the desired spin direction of the next shot (topspin or backspin).

Actions. Each action consists of two components: a latent code for MVAE to generate a kinematic target pose for the next frame, and joint corrections for the swing arm. The joint corrections include three Euler angles: two for the wrist joint (excluding the twist angle since the twist is limited for the wrist), and the twist angle for the elbow joint. The joint corrections overwrite the rotations from
the MVAE-produced pose, and the final corrected pose is used as the target pose for the low-level imitation policy to track.

5.3.2 Reward Function

We apply our framework to train control policies that enable the simulated character to hit an incoming tennis ball so that it bounces at desired location on the court (ball bounce position) and with a target spin direction. This objective is represented using two reward functions specified for stages before and after the racket-ball contact. Before contact, we apply the racket position reward $r^r_t$ to minimize the distance between the center of the racket head $\mathbf{x}^r_t$ and the ball position $\mathbf{x}^b_t$ when the character hits the ball (predicted motion phase $\theta_t$ gets close to $\pi$).

$$r^r_t = \exp(-\alpha_r ||\mathbf{x}^r_t - \mathbf{x}^b_t||^2) \cdot \exp(-\alpha_\theta ||\theta_t - \pi||^2), \quad (5.1)$$

where $\alpha_r$ and $\alpha_\theta$ are scaling factors. After contact, we apply the ball position reward $r^b_t$ to minimize the distance between the estimated ball bounce position $\hat{\mathbf{x}}^b$ and the target bounce position $\hat{\mathbf{x}}^b$ while ensuring the ball spins in the right direction as the target spin direction.

$$r^b_t = \begin{cases} 0 & \text{if } s^b \neq \hat{s}^b \\ \exp(-\alpha_\theta ||\hat{\mathbf{x}}^b - \hat{\mathbf{x}}^b||^2) & \text{if } s^b = \hat{s}^b \end{cases}, \quad (5.2)$$

where $s^b$ and $\hat{s}^b$ are binary variables that represent the simulated and target ball spin direction, respectively. A value of 1 denotes topspin (the ball spins forward) and a value of 0 denotes backspin (the ball spins backward). Since different strokes are needed to generate different spins to match $\hat{s}^b$, this reward causes the high-level policy to produce the appropriate stroke type even though the strokes in the source video are not annotated. At the moment of racket-ball contact, we immediately estimate $\hat{\mathbf{x}}^b$ and apply the same $r^b_t$ at every time step after contact. The two rewards are used differently at different curriculum stages described in the next section.

5.3.3 Training

We design the training strategy as follows. At the beginning of each episode, the character is initialized at a random court position near the baseline in a ready pose. The incoming balls are launched every 2-2.5 seconds from positions near the baseline of the opponent’s side of the court, with a launch velocity between 25-35 m/s, and a launch spin between 0-50 RPS. The ball can bounce anywhere between the service line and the baseline of the character’s side of the court, which covers a wide variety of incoming ball trajectories. To train the character to serve, we also initialize the character to a pre-service state and initialize the ball to be thrown into the air at the beginning of the training episode. The maximum episode length is set to be 300 frames (10 seconds) which allows
the character to practice four consecutive shots in each episode. We found that simulating multiple shots per episode leads to better performance compared to only one shot per episode.

**Curriculum Learning** To effectively and efficiently optimize the high-level policy, we adopt a curriculum that gradually increases the difficulty of the task over time. In the first stage of the curriculum, the objective is to quickly explore the motion embedding and control the character to move in the right direction so that the racket gets close to the incoming ball. Therefore, we train the policy only with the racket position reward $r_t$, and use a larger learning rate ($1 \times 10^{-4}$), higher action distribution variance $\Sigma$ ($0.25$), and a lower simulation frequency (120 Hz) for faster simulation. In the second stage of the curriculum, the goal is to control the racket so that the ball is hit over the net to the other side of the court. The target position is simplified as one of the three fixed positions at the left, center, and right of the court. In this stage, the policy is trained using both rewards with a higher weight on $r_t$ ($0.9$), and using a smaller learning rate ($2 \times 10^{-5}$), a lower $\Sigma$ ($0.04$). A higher simulation frequency (360 Hz) is also used by increasing the number of substeps in collision handling to ensure that racket-ball contact is simulated more accurately. Finally, the last stage of the curriculum encourages more precise control by sampling continuous target positions spanning the entire court. During this stage the policy is trained with an even smaller learning rate ($1 \times 10^{-5}$) and $\Sigma$ ($0.0025$).

### 5.3.4 Additional Kinematic Constraints

In addition to the wrist motion, other aspects of routine tennis motion may not be reconstructed accurately from video data. Most notably, (1) the character may not consistently keep their eyes on the ball and (2) the character’s non-dominant hand may not be gripping the racket during a two-handed swing.

While we could attempt to correct these errors by modifying the high-level policy to output corrections for more joints, and design specific rewards using domain knowledge of tennis, for simplicity we directly correct the kinematic motion with heuristics informed by knowledge of tennis.

**Keeping eyes on the ball.** In the real world, the player will rotate their head to keep their eyes on the ball. However, head/neck rotations are often poorly estimated by the kinematic pose estimator, leading to violations of this critical attribute of play. To correct head/neck motion, we first compute the offset angle between the head’s facing direction and the direction from the head to the ball’s current position and then add the offset angle back to the head/neck joints of the kinematic pose.

**Keeping both hands on the racket for two-handed backhands.** During a two-handed backhand swing, the player will hold the racket with both hands. However, pose estimation is typically
CHAPTER 5. LEARNING PHYSICS-BASED TENNIS CONTROL

Figure 5.5: The simulated character model is created from SMPL [53], with 24 rigid body segments and 72 DOF. The tennis racket is a combination of two solid cylinders and the grip is simplified by directly attaching the end of the racket handle to the wrist joint.

not sufficiently accurate to yield both hands tightly gripping the racket. To improve the visual realism of the generated motion, we adjust the kinematic pose to move the free hand close to the racket handle by solving inverse kinematics for the joints from the wrist to the shoulder along the arm of the free hand.

We remind the reader that as with the output of the high-level policy, motion changes due to kinematic constraints are converted into physically plausible motions after imitation by the low-level policy.

5.4 Results and Evaluation

We evaluate our learned controllers quantitatively in terms of their ability to successfully perform tennis tasks and via metrics that model the quality of the resulting motion. We also qualitatively evaluate the ability of our learned controllers to produce tennis motions that are humanlike and also reflect aspects of the style of motion demonstrated in source videos. We encourage the reader to view the supplementary video for demonstrations of the motion generated by our system.

5.4.1 Experimental Setup

The tennis racket is modeled as two solid cylinders with similar dimensions and masses as a real racket. The racket head is a rigid flat cylinder with a restitution of 0.9 and friction of 0.8 to simulate the effects of strings. We simplify the grip by directly attaching the end of the racket’s handle to the character’s wrist joint and model different grips by modifying the racket orientation relative to the character’s palm (Figure 5.5). The relative orientation between the palm and the racket is
set on a per-player basis according to the players’ grip style (“eastern” grip for Federer’s forehand, “semi-western” for Djokovic and Nadal).

All physics simulations are implemented using Issac Gym [55]. All policies are implemented as neural networks using PyTorch and trained using Proximal Policy Optimization (PPO) [75]. Advantage estimates for policy gradient updates are computed using the generalized advantage estimator GAE(λ) [74], and gradient updates are performed using the Adam optimizer [33]. In our experiments, the low-level policy is trained with approximately one billion samples, which requires around 12 hours on a single NVIDIA RTX A100 GPU. Training the MVAE requires four hours and the high-level policy is trained over five billion samples, which requires about two days. Unless otherwise stated, results in this section have residual force control enabled. Full implementation details can be found in Appendix A.

5.4.2 Metrics

To evaluate a simulated character’s task performance and motion quality, we consider the following quantitative metrics:

Task performance metrics. The character is initialized to a position at the center of the baseline and tasked to hit 15 consecutive random incoming tennis balls, which lasts about half a minute. The following statistics are then collected to evaluate the model’s task performance:

- **Hit rate**: the fraction of shots where the racket contacts the incoming ball.
- **Bounce-in rate**: the fraction of shots where the ball is hit and it bounces inside the court on the opposite side.
- **Bounce position error (bounce-pos err)**: average distance between the target bounce position and a shot’s actual bounce position when the ball lands inside the court.

Motion quality metrics. We also use the following metrics to measure the physical plausibility of the generated motions [101, 109]:

- **Jitter**: average of the third derivatives of all joint positions.
- **Foot sliding**: average displacement of body mesh vertices that contact the ground in two adjacent frames.

5.4.3 Learning Complex Tennis Skills

Using Federer’s motion data, we train a single controller to move the simulated character to the incoming ball, perform the appropriate swing, and hit the ball to a target location with the desired
Figure 5.6: Our simulated characters demonstrate diverse tennis skills that reflect coarse characteristics of the per-player video data they were trained on. (a)-(d) skills learned using Roger Federer’s motion data, who is a right-handed player and uses one-handed backhand. (e)-(f) skills learned using Novak Djokovic’s motion data, who is also a right-handed player but uses two-handed backhand. (g)-(h) skills learned using Rafael Nadal’s motion data, who is a left-handed player and uses two-handed backhand.

spin (topspin or backspin). Figure 5.6 illustrates examples of the diversity of shots generated by the controller, which includes serves, topspin forehand shots, topspin backhand shots, and backhand slices. Note how the controller learns to perform the appropriate swing motion (Figure 5.6b: topspin backhand vs. Figure 5.6d: slice backhand) to produce the target spin despite the lack of stroke annotations in the source video.

**Task performance.** To quantitatively evaluate task performance, we test the controller for 10K sessions (15 consecutive balls per session) and report statistics for all three task metrics in Table 5.1 (see row Fed-full). The controller is able to consistently hit incoming tennis balls despite their diverse trajectories (median hit rate: 0.92, median bounce-in rate: 0.85). The controller is also able to hit the ball near the specified target location. The median and mean bounce errors are both less than two meters, which is a level of capability similar to that of a skilled tennis player.

**Different player styles.** One of the advantages of learning skills from large-scale video data is that our system can learn per-player motion embeddings from video clips of each player, and then train different high-level policies for each embedding. We demonstrate this by training controllers from video clips of Nadal and Djokovic in addition to Federer. The three players have distinct playing styles: Federer and Djokovic are both right-handed, but Nadal plays left-handed. Federer
Table 5.1: Task performance of controllers learned from three players’ motions using our system. We show the 25%, 50%, and 75% quantiles using the metrics collected from 10K test sessions. (The controller is tasked to hit 15 incoming balls in a drill-like setting per session.) The learned controllers consistently hit a high fraction of balls back into the court, and achieve average bounce position errors of less than two meters.

<table>
<thead>
<tr>
<th></th>
<th>Hit rate</th>
<th>Bounce-in rate</th>
<th>Bounce-pos err (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fed-full</td>
<td>0.85/0.92/1.00</td>
<td>0.77/0.85/0.92</td>
<td>1.49/1.74/1.93</td>
</tr>
<tr>
<td>Djo-full</td>
<td>0.92/0.92/1.00</td>
<td>0.73/0.81/0.85</td>
<td>1.16/1.37/1.68</td>
</tr>
<tr>
<td>Nad-full</td>
<td>0.92/0.92/1.00</td>
<td>0.69/0.77/0.85</td>
<td>1.31/1.56/1.89</td>
</tr>
</tbody>
</table>

Table 5.2: Ablations on the effect of physics correction (PhysicsCorr) and hybrid control (Hybrid-Ctr). Removing either component of the system results in decreased task performance.

<table>
<thead>
<tr>
<th></th>
<th>Hit rate</th>
<th>Bounce-in rate</th>
<th>Bounce-pos err (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o PhysicsCorr</td>
<td>0.85/0.92/0.92</td>
<td>0.69/0.77/0.85</td>
<td>2.00/2.37/2.81</td>
</tr>
<tr>
<td>w/o HybridCtr</td>
<td>0.69/0.85/0.92</td>
<td>0.31/0.46/0.54</td>
<td>2.82/3.43/4.00</td>
</tr>
<tr>
<td>Fed-full</td>
<td>0.85/0.92/1.00</td>
<td>0.77/0.85/0.92</td>
<td>1.49/1.74/1.93</td>
</tr>
</tbody>
</table>

uses a one-handed backhand, while Djokovic and Nadal use two-handed backhands. Qualitatively, as shown in Figure 5.6, learned skills capture the coarse attributes of a player’s style (handedness and whether they use a one or two-handed backhand). We report task performance for the controllers trained on Djokovic and Nadal videos in Table 5.1 (Djo-full and Nad-full). Our system learns high performing controllers in all three cases.

Tennis rallies between two players. Although the controllers are trained in a single-player setting (i.e., a single simulated character without an opponent), once trained, they can be directly applied to a two-player simulation. Specifically, we use two trained controllers (of the same player or different players) to drive two simulated characters to play tennis rallies against each other. We were able to simulate a rally of 38 shots using the two controllers Fed-full and Djo-full, lasting for 41 seconds and ending with a miss by Fed-full.

5.4.4 Tackling Low-Quality Demonstrations

A key challenge in this work is learning from low-quality motion data extracted from videos. We conduct two ablation studies to show the effectiveness of our proposed solutions.

Constructing motion embedding. Our system leverages two key steps to process the noisy motions \( M_{\text{kin}} \) estimated by the kinematic pose estimator into smooth and plausible motions \( M_{\text{vae}} \).
CHAPTER 5. LEARNING PHYSICS-BASED TENNIS CONTROL

Table 5.3: Motion quality evaluation. We compare the motion output by our full system (Fed-full), the motion from the ablation (w/o PhysicsCorr), and motions at different stages of our system: estimated kinematic motion (M_{kin}), physically corrected motion (M_{corr}), and motion output by MVAE (M_{vae}). The motion generated from our full system shows higher motion quality (less jitter and foot sliding) than w/o PhysicsCorr and the motions at intermediate stages.

<table>
<thead>
<tr>
<th></th>
<th>Jitter (10³ m/s³)</th>
<th>Foot sliding (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_{kin}</td>
<td>6.08</td>
<td>7.41</td>
</tr>
<tr>
<td>M_{corr}</td>
<td>3.14</td>
<td>1.46</td>
</tr>
<tr>
<td>M_{vae}</td>
<td>0.96</td>
<td>4.70</td>
</tr>
<tr>
<td>w/o PhysicsCorr</td>
<td>1.19</td>
<td>2.82</td>
</tr>
<tr>
<td>Fed-full</td>
<td><strong>0.51</strong></td>
<td><strong>1.46</strong></td>
</tr>
</tbody>
</table>

First, the noisy motions in M_{kin} are corrected by the low-level imitation policy using physics simulation. Second, the corrected motions M_{corr} are further denoised by training MVAE to embed the motion into a smooth motion space. Table 5.3 shows the motion quality of the motions at different stages. The physics-corrected motion M_{corr} exhibits less jitter and foot sliding, and the motions generated by the MVAE (M_{vae}) are even more smooth. As a side-effect, the smoothing by MVAE also increases foot sliding in M_{vae}, but these artifacts are largely removed by the final step of imitating M_{vae} with physics simulation (Fed-full). To further evaluate the impact of the physics-based correction on the learned controllers, we can train the MVAE using the original outputs of the pose estimator M_{kin}, and then use the resulting MVAE to train the high-level policy (w/o PhysicsCorr). Table 5.3 shows that the controller trained without physics-based correction produces motion with more jitter and foot sliding compared to Fed-full. Table 5.2 also shows that the task performance of the controller trained without correction also decreases, especially in bounce position error, which indicates the importance of using physics-based correction to construct a good motion embedding.

**Hybrid control for wrist motion** To evaluate the effectiveness of the proposed hybrid control for the wrist motion, we can train the high-level policy to only predict the latent code for the MVAE and use the motion output from the MVAE without joint corrections as the target kinematic motion (w/o HybridCtr). As shown in Table 5.2, although the agent is still able to achieve a reasonable hit rate, the bounce-in rate drops nearly by half and bounce position error increases significantly. This indicates that the proposed hybrid control is essential for achieving the challenging task of returning the ball close to the target location.
5.4.5 Analysis of One Million Simulated Shots

To further understand the performance of our learned controllers, we simulate a million shots using Fed-full and analyze the performance metrics conditioned on certain features of the shots.

Incoming ball’s velocity and spin We plot the hit rate conditioned on the incoming ball’s velocity and spin when launched, shown in Figure 5.7(a). Faster incoming balls are missed more often and balls with faster spin are also more difficult to hit, which is consistent with real-world tennis.
Table 5.4: Ablation of various design choices of our system. The table provides average metrics collected from 10K test sessions. All design decisions contribute to the task performance of the controller.

<table>
<thead>
<tr>
<th></th>
<th>Hit rate</th>
<th>Bounce-in rate</th>
<th>Bounce-pos err (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o Root</td>
<td>0.61</td>
<td>0.45</td>
<td>2.28</td>
</tr>
<tr>
<td>w/o Phase</td>
<td>0.17</td>
<td>0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>w/o FutureObs</td>
<td>0.86</td>
<td>0.30</td>
<td>4.51</td>
</tr>
<tr>
<td>w/o EstBounce</td>
<td>0.86</td>
<td>0.42</td>
<td>3.70</td>
</tr>
<tr>
<td>w/o Curriculum</td>
<td>0.89</td>
<td>0.71</td>
<td>2.67</td>
</tr>
<tr>
<td>Fed-full</td>
<td><strong>0.89</strong></td>
<td><strong>0.81</strong></td>
<td><strong>1.73</strong></td>
</tr>
</tbody>
</table>

Incoming ball’s bounce position In Figure 5.7(b) and (c), we plot the hit rate and average bounce position error conditioned on the incoming ball’s bounce position. We observe that balls that bounce shorter in the middle of the court are easier shots to hit, while balls that bounce close to the edges of the court are much more challenging (consistent with real tennis). A similar trend is also evident in Figure 5.7(c). Interestingly, incoming balls that bounce on the right side of the court are returned with lower bounce position error, suggesting that our right-handed simulated character has better control with forehand shots.

Reaction distance Reaction distance is the distance that the character must move to reach an incoming ball. As shown in Figure 5.7(d) and (e), balls that require a longer reaction distance result in shots with a lower bounce-in rate and higher bounce position error. These balls are more difficult to reach and leave the character with less time to adjust.

5.4.6 Ablation Studies

We conduct additional ablations to isolate the importance of key design choices to the overall task performance. The results are summarized in Table 5.4. First, we observe that both adaptions to the original MVAE are crucial. Omitting the conditioning on global root position (w/o Root) results in global drifts of the kinematic motion due to error accumulation in autoregressive generation. The result is that the simulated character fails to hit subsequent shots. Further, without predicting the motion phase (w/o Phase), we are not able to minimize the distance between the racket and the ball at the ball contact time of the swing motion, i.e., the character can run into the ball at any time. The result is that the simulated character rarely hits the ball. Second, we find that it is important to estimate the ball trajectory in the future. In one ablation, we use historical ball positions instead of estimated future ball positions as the observation of the high-level policy (w/o FutureObs). Using historical ball positions results in a significant decrease in bounce-in rate,
CHAPTER 5. LEARNING PHYSICS-BASED TENNIS CONTROL

5.4.7 Sensitivity to Database Size

We also study the impact of input video database size on task performance and output motion quality (Figure 5.8). We retrain the motion embedding and the high-level policy using 12.5%, 25%, 50%, and 75% of the motion data for Federer. Figure 5.8(a) shows that using increasing motion data to construct the motion embedding improves task performance, suggesting the value of acquiring large amounts of motion data from large scale video datasets. With only 12.5% of data (about ten minutes of motion), the hit rate and bounce-in rate are significantly reduced because the learned motion embedding is not dense enough for successful high-level motion planning. Additionally, with less motion data available to model the latent space, the smoothing effect of the embedding is reduced due to decreased signal to noise ratio in reconstructing the repetitive motion, leading to more jitter in the decoded motion and the final simulated motion (Figure 5.8(b)).
5.4.8 Inspection of Joint Torques

To evaluate whether the joint torques in our simulated motions are physically plausible comparable to the range of real humans, we inspect the joint torques generated by the PD controller when controlled by Fed-full. We simulate one minute of motion and draw the box plot for each joint in Figure 5.9. As shown in the figure, the median values for most of the joints are comparable to those of real humans, but the maximums are beyond the limits of real humans. The major concern of the current controller is that no torque limit is enforced, leading to unreasonably large torques. We would like to set torque limits in the future work and expect it to encourage more physically plausible motions.

5.4.9 Effects of Removing Residual Force Control

Our system enables residual force control to improve the tracking performance of the low-level imitation policy. However, residual forces may also lead to unrealistic motion and cannot be implemented in a real-world mechanical system. As described in Section 3.3, our system optionally allows for the development of the imitation policy without residual force control. Table 5.5 evaluates the overall effect of removing residual force control on task performance and motion quality. Without residual force control, foot sliding is reduced by 40% and the resulting motions are perceptibly more plausible (see supplemental videos). However, the increased motion quality comes at the cost of a 12% reduction in hit rate and 15% decrease in bounce-in rate.

To better understand the differences and the limit of the controllers developed with/without residual force control, we conduct the following experiment. We curated a list of control problems,
Table 5.5: Removing residual force control yields more realistic motion (40% reduction in foot sliding) but reduces the tracking ability of the low-level imitation policy, resulting in reduced overall task performance. Users can select whether or not to employ residual forces based on desired performance-motion quality needs.

<table>
<thead>
<tr>
<th></th>
<th>Hit rate</th>
<th>Bounce-in rate</th>
<th>Bounce-pos err (m)</th>
<th>Foot sliding (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o RF</td>
<td>0.78</td>
<td>0.69</td>
<td>1.85</td>
<td>0.87</td>
</tr>
<tr>
<td>Fed-full</td>
<td><strong>0.89</strong></td>
<td><strong>0.81</strong></td>
<td><strong>1.73</strong></td>
<td>1.46</td>
</tr>
</tbody>
</table>

Table 5.6: Task performance of w/o RF and Fed-full when tested with a list of curated control problems with increased reaction distance (Y for successful hit and bounce-in). Fed-full manages to move to reach the incoming balls when the reaction distance is large while w/o RF fails.

<table>
<thead>
<tr>
<th>Reaction distance</th>
<th>1 m</th>
<th>2 m</th>
<th>3 m</th>
<th>4 m</th>
<th>5 m</th>
<th>6 m</th>
<th>7 m</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o RF</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Fed-full</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

where the character is initialized at the same court position and the ball is coming to their right side (forehand) with increased reaction distance from 1 m to 7 m. Table 5.6 shows the task performance of the control problems when tested with w/o RF and Fed-full. It shows that when reaction distance is large which requires the character to react faster to reach the ball, Fed-full is capable of controlling the character to run faster and move to reach the ball while w/o RF fails. The difference is more obvious when we compare the character’s root velocity in the generated motions (Figure 5.10). The maximum root velocity is above 5 m/s when controlled by Fed-full, while it is below 4 m/s when controlled by w/o RF. The further inspection of the generated motions suggests that w/o RF struggles to track faster running reference motion closely and may fall down at the extreme cases (Figure 5.12 top row).

In Figure 5.11 we also plot the magnitude of residual forces and torques applied at each step when tested with increased reaction distance by Fed-full. First, it shows that the required residual forces/torques are smaller when the characters do not need to run very fast. However, the amount of residual forces/torques quickly explode when tracking fast running reference motions. In particular, the peak occurs when the characters just try to recover balance after a fast reaction to hit the ball, which matches with our observation of the unnatural recovery behavior at the extreme cases. When the character is recovering from the state where almost all weights are on the striking side, instead of slowly transferring weight to the other side, the character seems to effortlessly tip-toe through this phase as if they are attached to some invisible spring (Figure 5.12 bottom row).

In summary, our system provides users with the flexibility to choose the desired scale of residual
CHAPTER 5. LEARNING PHYSICS-BASED TENNIS CONTROL

Ball contact time

Figure 5.10: Root velocity of the simulated characters under different reaction distances (RD), when controlled by Fed-full (left) and w/o RF (right). The simulated character runs faster with residual force control enabled. The maximum root velocity is above 5 m/s when controlled by Fed-full, while it is below 4 m/s when controlled by w/o RF.

forces based on their motion quality and task performance needs. Future work will explore ways to improve task performance without relying on residual force control.

5.4.10 Limitations

We note that although our per-player controllers successfully reflect gross aspects of each player’s style, there remains much work to be done to accurately reproduce more nuanced details of professional-level tennis footwork and swings. For example, racket head velocity, and correspondingly the length of swing follow through, is shorter in the generated motions than the real-life examples. Wrist pronation, and thus racket head position during the back swing, is not faithfully reproduced. Also, our generated motions fail to capture how players place their non-swinging hand on the throat of
Figure 5.12: Failure cases. Top: the character simulated without residual force control may fall down when tracking fast running reference motion. Bottom: the character simulated with residual force control may exhibit implausible balance when running fast to reach the ball and trying to recover from the swing motion.

The racket during swing preparation. To more accurately reproduce these details of professional athlete performance, higher-fidelity motion examples and improved simulation fidelity (more accurate models of human anatomy and ball-string contact) are likely necessary.
Chapter 6

Improving Fine-Grained Motion Details

Although we are able to replicate the coarse techniques of the players from their videos, such as hitting the ball with right hand vs. left hand or holding the racket with one hand vs. two hands, the fine-grained motion details of the original performance are not yet reproduced by the controller described in Chapter 5. For instance, players actively use pronation of the forearm when hitting forehands such as to create even more spin when hitting the ball. Additionally, players should be accelerating the racket at the beginning of the stroke and then let it swing with full momentum through the ball, finishing high up over the opposite shoulder/on the opposite side of their body. As shown in Figure 6.1, neither of the styles are present in the results produced by the controller.

The fundamental reason lies in the fact that the wrist and racket motion can not be accurately estimated from broadcast videos due to heavy motion blur, as discussed in Section 3.3.5. On the other hand, it is possible to obtain a small amount of high-fidelity motion examples, such as from slow-motion recordings of the practice shots of these players. Professional players are strictly trained to perform almost the same skill, making these videos valuable source for complementing the missing motion details from broadcast videos. These videos are in high resolutions, focusing on the player, and they are also in high frame rate (240 fps), making vision-based estimation to produce more accurate motion. In particular, we are able to annotate the racket position precisely from these videos, which provide a strong constraint for solving the wrist motion via inverse kinematics (IK). More importantly, the multi-view input provides additional constraints when the racket is occluded or suffers from monocular ambiguity.

In this chapter, we will describe the approach for reconstructing high-quality motions from the slow-motion videos and the solution to incorporate the sparse motion examples into the physics-based controller described in Chapter 5. Our preliminary results demonstrate that it is possible to
improve fine-grained motion details leveraging the pristine but sparse motion data, such as increasing the length of the follow-through motion.

6.1 Approach

We start by following the video annotation pipeline in Chapter 3 to annotate the slow-motion videos. In order to get more accurate wrist motion, we also annotate the 2D racket positions and use them as constraints to solve the wrist motion via optimization. Finally, the reconstructed high-quality motion is used to correct the fine-grained motion details in the motion embedding by fine-tuning the MVAE.

6.1.1 Video Annotation

For any swing skill of one player (e.g. Federer's forehand topspin), we collect N instances of slow-motion recordings from a variety of camera angles. We show a few examples in Figure 6.2.

For each swing instance, we manually segment the video clip of 2 seconds centered at the ball contact frame, referred as $c_i$. Each clip has a length of $T = 480$ frames. We use HybrIK [44] to estimate the joint rotations at all frames $t$ for each sequence $c_i$, denoted as $q_i(t)$. The global root orientation/trajectory estimation from Chapter 3 can not be employed because the mapping between the court space and screen space can not be reliably established under the cropped view of the court. Instead, we set the global root orientation assuming the player is facing forward at the beginning of the clip (in ready pose). Since our focus of this motion is on the upper body, we simply set the global root position of the kinematic motion to be static throughout the sequence.

Although the high resolution and frame rate makes it easier to estimate the joint positions/rotations in general, estimating accurate wrist motion remains a challenge for two reasons. First, the wrist joint is next to the leaf joint (hand) along the kinematic tree on the arm. Besides, the short distance
between the wrist and hand joint makes it an ambiguous problem for the IK solver. Second, the publicly available mocap datasets used for training HybrIK and other vision-based pose estimation approaches contain less diverse motion of the wrist joint. As a result, the predicted wrist motion is biased toward the default wrist configuration (0 degrees for all three dofs).

Since it is difficult to directly estimate the wrist motion, we can instead extend the kinematic tree from the wrist joint to the racket head center, which adds constraint for solving the wrist rotations via IK. Thanks to the high frame rate, it is possible to annotate the racket precisely in each frame. We manually annotate the 2D position of the racket center in each frame, denoted as $\tilde{x}_{rc}^i(t)$. In order to solve the accurate wrist twist angle, we can additionally annotate the 2D positions of the two side points of the racket (points at the same height as the racket center) to determine the racket orientation.

### 6.1.2 Wrist Optimization

Given the estimated joint rotations $q_i^t(t)$, estimated camera projections $H^c_i(t)$ (camera space to screen space) and annotated 2D racket positions $\tilde{x}_{rc}^i(t)$ for all the $N$ clips, we solve the wrist rotations $q_w^i(t)$ by forming the following optimization problem jointly across different viewpoints:

$$
\min_{\{q_i^w(t)\}_{t=1}^{T}} \alpha_d \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{w_i}{w_t} \sum_{t=1}^{T} \|H^c_i(t)x_{rc}^i(t) - \tilde{x}_{rc}^i(t)\|^2 + \alpha_s \sum_{t=1}^{T-1} \|q_i^w(t) - q_i^w(t+1)\|^2. \quad (6.1)
$$

The goal of this optimization is to jointly minimize the re-projection loss of the racket positions and the smoothness of the wrist angles across neighboring frames. The 3D racket positions $x_{rc}^i(t)$ in camera space can be computed from $q_i^t(t)$ via forward kinematics.

Although the optimization is conducted upon all the different viewpoints to address the ambiguities and occlusions, there still exist differences in the swing motion across the $N$ instances, such as hitting a low ball vs. high ball. To account for the differences across instances, we solve $q_i^w(t)$ by running this optimization for each clip $c_i$, with higher weight $w_i$ over the re-projection loss of $c_i$. In order to add constraints of the two sides points, we can add additional re-projection loss by replacing the racket center to the side points. The optimization is implemented in Pytorch using Adam optimizer. We use step size of 0.001 and run the optimization for 10k steps. We set $\alpha_d = 1$.
CHAPTER 6. IMPROVING FINE-GRAINED MOTION DETAILS

Figure 6.3: Comparison between reconstructed motions from broadcast videos (top) and slow-motion videos (bottom). The yellow and pink characters visualize the kinematic estimated motion and the simulated motion from the low-level imitation policy. Note that the fine-grained motion details such as pronation (first column) and longer follow-through (last column) are neither captured in the kinematic motion or the simulated motion when reconstructed from broadcast videos. However, these details are better preserved when reconstructed from slow-motion videos.

\[ \alpha = 0.01 \]

Once we obtain the final kinematic motions, we downsample them into 30 fps and run the imitation policy (Section 3.3) to track the kinematic motions and store the simulated motions as the high-quality tennis motion dataset, referred as \( M_{hq} \). Figure 6.3 shows comparison of two forehand swings between the motion reconstructed from broadcast videos and slow-motion videos, as well as the frames of the original broadcast video. It is obvious that the reconstructed motion from slow-motion videos shows improved pronation and follow-through.

6.1.3 Motion Blending

Although there could be other solutions to make use of the high-quality motions \( M_{hq} \), we choose the most straightforward extension given the framework introduced in Chapter 5, which is to inject \( M_{hq} \) into the motion embedding learned from broadcast videos. In particular, we would like to use \( M_{hq} \) to correct the motion embedding so that the output motion \( M_{vae} \) shows improved fine-grained details upon the upper body motion during swing while remains diverse motions of footwork during swing and transitions.

We propose a solution to fine-tune the MVAE from Section 5.2 using both the motion from
Chapter 6. Improving Fine-Grained Motion Details

Figure 6.4: Comparison between motions generated by controllers learned using the old motion embedding (top) and the corrected motion embedding (bottom). The motion generated by Fed-HQ shows more clear wrist pronation before contact (a), more wrist bend during the swing (b)(c), and a longer follow-through after contact (d).

6.2 Results and Evaluation

In this section, we will show our preliminary results to demonstrate that it is possible to improve fine-grained motion details leveraging the pristine but sparse motion data. We collect $N = 10$ slow-motion clips of Federer’s forehand topspin to conduct our experiments and the learned controller is referred as Fed-HQ.

6.2.1 Qualitative Evaluation

In Figure 6.4, we compare the motions generated by controllers learned using the old motion embedding (Fed-full) and the corrected motion embedding (Fed-HQ). We find that the corrected motion embedding effectively blends the two motion dataset, where the swing motion now resembles that of the pristine motion, while the foot work is learned from the broadcast videos. In particular, it shows more clear wrist pronation before contact, and a longer follow-through after contact, which is supposed to occur in a professional swing to slow down the racket.
Table 6.1: Quantitative evaluation of improved style after integration of high-quality motions. The improved style also brings increase in both racket velocity and the shot velocity/spin after contact, which are closer to professional performance.

<table>
<thead>
<tr>
<th></th>
<th>Shot velocity (m/s)</th>
<th>Shot spin (RPS)</th>
<th>Racket max velocity (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fed-full</td>
<td>26.3</td>
<td>31.6</td>
<td>12.5</td>
</tr>
<tr>
<td>Fed-HQ</td>
<td>29.2 (11%↑)</td>
<td>42.4 (17%↑)</td>
<td>13.3 (7% ↑)</td>
</tr>
</tbody>
</table>

### 6.2.2 Quantitative Evaluation

We also conduct quantitative evaluation using the following metrics:

- **Shot velocity**: the magnitude of ball velocity at launch time.
- **Shot spin**: the magnitude of ball spin at launch time.
- **Racket maximum velocity**: the maximum velocity of racket head center during swing.

Table 6.1 shows the metrics averaged over 10k forehand shots between the two controllers. The improved style also brings increase in both racket velocity and the shot velocity/spin after contact, which are closer to professional performance.
Chapter 7

Modeling Player Behaviors

In the previous chapters, our focus is on addressing the tennis control problem, which is aimed to synthesize realistic motions that satisfy the input control constraints. We have not yet talked about how to specify these constraints, other than random generation (e.g., randomly generated target shot placement positions). However, generating these high-level control inputs is also a crucial problem because creating realistic virtual characters is not only about making them move realistically, but also have them make reasonable behavioral decisions, such as what type of stroke to use and where to place the next shot. Fortunately, the broadcast tennis videos also provide a rich source of information about how athletes play their sport (tendencies, strengths, and weaknesses). In this chapter we take inspiration from the field of sports analytics, which uses analysis of sports videos to create predictive models of athlete behavior. However, rather than use these models to inform coaching tactics or improve play, we combine sports behavioral modeling with the tennis controllers we have developed to synthesize animations that both move and behave like professional tennis athletes.

Our approach takes as input the database of broadcast tennis videos, annotated with important match play events (e.g., time and location of ball contact, type of stroke), as described in Chapter 3. The database contains a few thousand shots from each player, which we use to construct behavior models of how the player positions themselves on the court and where they are likely to hit the ball in a given match play situation.

The fundamental challenge is to build a model that produces realistic behavior decisions to a wide range of competitive point play situations, given only a few thousand shots. To generate realistic output from limited inputs, we employ domain knowledge of tennis – the shot-cycle state machine described in Section 2.1 to constrain and regularize the player behavior modeling. Furthermore, we use simple non-parametric models conditioned on a low-dimensional, discretized representation of point state instead of data hungry models such as neural networks.

We demonstrate that these behavioral models can provide control inputs to the tennis controllers so that the synthesized tennis animations are “realistic” both at a fine-grained level in that they
Figure 7.1: We build player behavior models using player and ball trajectories extracted from video to predict a player’s shot placement (where they hit the ball to) and where they position themselves on the court against a specific opponent, which can be used as the input constraints for the tennis control problem. As a result, the synthesized tennis points share similar style and strategy as their real-world counterparts. Visualizations depict where our model of Novak Djokovic hits the ball to (shot placement) and positions himself (recovery position) when hitting backhands against Roger Federer and Rafael Nadal. Djokovic’s strategy changes to hit to the right side of the court vs. Federer (his weaker backhand side) while a majority of shots vs. Nadal go to the left.

Successfully achieve the task goals and move naturally like human, but also at a macro level in that they capture real-life strategies (hitting the ball to an opponent’s weakness) and tendencies (aggressive vs. defensive court positioning) during point play. An evaluation with expert tennis players shows that the rallies generated using the behavior models are significantly more realistic in terms of player behavior than the ones generated via controllers that only solve the control problem.

7.1 Related Work

The ability to capture accurate player and ball tracking data in professional sports venues [60, 76, 80] has spurred interest in aiding coaches and analysts with data-driven predictive models of player or team behavior. Recent work has used multiple years of match data to predict where tennis players will hit the ball [18, 91], whether a point will be won [90], how NBA defenses will react to different offensive plays [38], and analyze the risk-reward of passes in soccer [68]. We are inspired by these efforts, and view our work as connecting the growing field of sports play forecasting with the computer graphics challenges of creating controllable virtual characters.

7.2 Approach

Using the database of annotated shot cycles, we construct statistical player behavior models that input the state of the point at the beginning of the shot cycle, and produce player shot selection and recovery position decisions for the shot cycle. We denote this set of behavior decisions as \( G^* = (c^*, v^b_0, x^b_0, x^r_0) \). (We use * to denote behavior model outputs.) The first three components
CHAPTER 7. MODELING PLAYER BEHAVIORS

of $G^*$ refer to shot selection decisions: shot type $c^*$ (forehand or backhand, topspin or underspin), shot velocity $v^*$ (the magnitude of average ground velocity from ball contact to bounce) and shot placement position $x^*_b$ (where the shot should bounce on the opponent’s side of the court). The last component, $x^*_r$, denotes the player’s recovery position goal.

Tennis point play yields a large space of potential point configurations that are only sparsely sampled by shot cycle examples in our database. This prevented use of data hungry models (e.g., neural nets) for modeling player behavior. To generate realistic behaviors (follow player specific tendencies, obey tennis principles, model important rare actions) for a wide range of situations, we use simple non-parametric models conditioned on a low-dimensional, discretized representation of point state. We specialize these models and sampling strategies for the different components of player behavior.

7.2.1 Shot Selection

The shot selection behavior model generates $(c^*, v^*_b, x^*_b)$ in $B^*$. In addition to the positions of both players at the start of the shot cycle and the trajectory of the incoming ball, the shot selection behavior model also utilizes the following estimated future state features that influence decision making during tennis play.

*Estimated ball contact time/position.* We estimate the time and court position where the player will contact the incoming ball by determining when the ball’s trajectory intersects the plane (parallel to the net) containing the player. This computation assumes that a player moves only laterally to reach the ball (not always true in practice), but this rough estimate of the contact point is helpful for predicting shot placement and recovery behavior.

*Velocity to reach estimated contact position.* Rapid movement to reach a ball can indicate a player under duress in a rally, so we compute the velocity needed to move the player from their position at the start of the shot cycle to the estimated ball contact position.

*Opponent’s estimated position at time of contact.* A player’s shot selection decisions depend on predictions of where their opponent will be in the future. The opponent’s most likely position at the time of player ball contact is given by the recovery position of the opponent’s current shot cycle. We use the method described in Section 7.2.2 to estimate this position.

We construct a point state descriptor $D_s$ by discretizing the following five features: the estimated positions of both players at time of contact, the incoming ball’s starting and bounce position, and the magnitude of player velocity needed to reach the estimated contact point. We discretize the magnitude of player velocity uniformly into five bins. Figure 7.2 visualizes the 2D court partitioning used to discretize the estimated positions of both players at ball contact and the incoming ball’s starting and bounce positions. We divide the region between the singles sidelines uniformly into deuce, center, ad regions, and place the front/back court partitioning line halfway between the service and baseline since player movement toward the net past this point typically indicates a transition
to hitting volleys. The choice of six regions balances the need for sufficient spatial resolution to capture key trends (like cross court and down-the-line shots, baseline vs. net play) with the benefits of a low-dimensional state representation for fitting a behavior model to sparse database examples. Overall, this discretization yields 1080 unique point states. (See Appendix A for details on the discretization of all features.)

**Model construction** As an offline preprocess, we use the clip database to estimate statistical models conditioned on $D_s$. For each unique state, we identify all database clips that begin from that state, and use these data points to estimate models for the various shot selection decisions. (When computing $D_s$ for database clips, we directly use the known player positions at ball contact instead of estimating these quantities.) We model shot type as a categorical distribution $p(c|D_s)$ and model shot velocity, $p(v_b|D_s, c)$, and shot placement, $p(x_b|D_s, c)$, using 1D and 2D kernel density estimators (KDE) [77] conditioned on both $D_s$ and shot type. (Figure 7.2 visualizes the $p(x_b|D_s, c = \text{FH-T})$ in red). As many point state descriptors correspond to at most a few database clips, we use a large, constant Gaussian kernel bandwidth for all KDEs. (We use Scikit-learn’s leave-one-out cross-validation to estimate the bandwidth that maximizes the log-likelihood of the source data [62].)
Since a player’s shot placement depends both on their own playing style and also their opponent’s characteristics (e.g. a player may be more likely to hit to their opponent’s weaker side), we build opponent-specific models of player behavior by limiting source data to matches involving the two players in question.

Model evaluation To generate a player’s shot selection decision during a novel rally, the system computes $D_s$ from the current state of the point, then samples from the corresponding distributions to obtain $(c^*, v_b^*, x_b^*)$. To emphasize prominent behaviors, we reject samples with probability density lower than one-tenth of the peak probability density. If the generated shot placement position falls outside the court, the player’s shot will be an error and the point will end. Therefore, the shot selection model implicitly encodes the probability that a player will make an error given the current point conditions.

The approach described above estimates behavior probability distributions from database clips that exactly match $D_s$. When there is insufficient data to estimate these distributions, we relax the conditions of similarity until sufficient database examples to estimate a distribution are found. Specifically, we estimate the marginal distribution over an increasingly large set of point states, prioritizing sets of states that differ only in features (variables in $D_s$) that are less likely to be important to influencing behavior. We denote $D_s^k$ as the set of point states that differ from $D_s$ by exactly $k$ features. Our implementation searches over increasing $k$ ($1 \leq k \leq 4$) until a $D_s^k$ is found for which there are sufficient matching database clips (at least one clip in our implementation). For each $k$, we attempt to marginalize over features in the following order determined from domain knowledge of tennis play: velocity to reach the ball (least important), incoming ball bounce position, incoming ball starting position and lastly the opponent’s position at time of contact (most important). In practice, we find 91% of the shot selection decisions made in novel points do not require marginalization.

7.2.2 Player Recovery Position

A player’s recovery position reflects expectations about their opponent’s next shot. Since a player’s shot selection in the current shot cycle influences their opponent’s next shot, the input to the player recovery position behavior model is a new descriptor $D_r$ formed by concatenating $D_s$ (the current state of the point) with the shot placement position (from the shot selection behavior model).

One major choice in recovery positioning is whether the player will aggressively approach the net following a shot. We model this binary decision $Y$ as a random variable drawn from a Bernoulli distribution $p(Y|D_r)$, where probability of $Y=1$ is given by the fraction of database clips matching $D_r$ that result in the player approaching the net (recovering to front regions of the court). We model $p(x_r|D_r, Y)$ by constructing a 2D KDE from the recovery position of database clips matching both $D_r$ and $Y$. (Figure 7.2 visualizes $p(x_r|D_r, Y = 0)$ in blue.) When insufficient matching database clips exist, we marginalize the distribution using the same approach as for the shot selection behavior.
model.

In contrast to shot selection, where we sample probability distributions to obtain player behavior decisions (players can make different shot selection decisions in similar situations), the player behavior model designates the player’s recovery position goal as the most likely position given by \( p(x_r|D_r,Y) \). This reflects the fact that after the strategic decision of whether or not to approach the net is made, a player aims to recover to the position at the baseline (or the net) that is most likely to prepare them for the next shot.

### 7.3 Results and Evaluation

We utilize the Wimbledon dataset introduced in Chapter 3 to construct the player behavior models for the three male players: Federer, Nadal and Djokovic. These models are then integrated into the kinematic tennis controller described in Section 4.3 to synthesize novel tennis points. Our analysis and user study of the synthesized tennis points show that they capture real-life strategies (hitting the ball to an opponent’s weakness) and tendencies (aggressive vs. defensive court positioning) during point play.

#### 7.3.1 Emergent Player Behavior

Our player behavior models produce points that echo real-world player tendencies and advanced tennis play. To evaluate emergent behavior, we generate novel points using the three male tennis players in our database (200 points for each of the three pairings: Djokovic vs. Federer, Djokovic vs. Nadal, Federer vs. Nadal). We summarize our observations from these synthesized points here, but refer the reader to the supplementary video for a detailed inspection of results. (Note that in this section, mentions of a specific player’s behavior refer to the actions of virtual characters in synthesized points, not the behavior of the real-life player as observed in database clips.)

**Recreation of player style.** Our behavioral models capture elements of player-specific style that are consistent with well-known tendencies of the real-life players. For example, on average Federer stands one meter closer to the baseline than Djokovic and Nadal, suggestive of his attacking style (Figure 7.3-c). Federer also hits forehands with significantly higher ground velocity than backhands, while his opponents exhibit greater balance between their forehand and backhand sides. Notice how on average, all players recover to an off-center position on the court (right-handed Federer and Djokovic to the left side, and left-handed Nadal to the right). This positioning makes it more difficult for their opponents to hit shots to their weaker backhand side. Cross-court shots are the most common shots by all players in the synthesized rallies (Figure 7.3-a/b), echoing statistics of real play (cross court shots have lower difficulty). Nadal hits a particularly high percentage of his
CHAPTER 7. MODELING PLAYER BEHAVIORS

Figure 7.3: Shot placement (forehand/backhand) and recovery positions of Federer (right-handed), Djokovic (right-handed), and Nadal (left-handed) in synthesized points. (The red and blue heat maps visualize a KDE fit to the results of simulation, not the distribution used by our behavior models.) In general a majority of shots are hit cross court, except for Nadal’s backhand which is distributed equally around the court. Federer stands nearly one meter closer to the baseline (only 0.9 m behind baseline) than Djokovic (1.6 m) and Nadal (1.9 m), suggesting more aggressive play. All players recover to an off-center court position, making it harder for their opponents to hit the ball to their backhand side.

Forehand cross court. Conversely, Nadal’s backhand shot placement is the most equally distributed to both sides of the court.

**Playing to opponent’s weaknesses.** In competition, a player’s behavior is heavily influenced by the strengths and weakness of their opponent. Figure 7.1 illustrates differences in Novak Djokovic’s shot placement when hitting from the left side of the court when playing Roger Federer (left) and Rafael Nadal (right). Djokovic places most of his shots cross court to Federer’s weaker backhand side. In contrast, when playing left-handed Nadal, Djokovic places a majority of his shots down the line to Nadal’s backhand. Videos generated using these models reflect these decisions. For example, Djokovic-Federer videos contain many cross-court rallies. Figure 7.4-a shows how Nadal plays Federer and Djokovic differently even though both opponents are right handed. Nadal avoids Federer’s exceptionally strong forehand side (left side of the figure), but is more willing to hit the ball to Djokovic’s forehand.

**Players demonstrate good tennis principles.** Compared to random play, the virtual characters play the game according to good tennis principles. As is the case in real play, the shot placement of virtual characters is affected by whether they are under duress in a rally. For example, Djokovic’s shot placement changes against Nadal, depending on how fast Djokovic must run to the right to hit the incoming cross-court ball (Figure 7.4-b). When Djokovic must run quickly because he is out of position (left panel), he typically hits a safer (defensive) cross-court shot. However, when Djokovic is well positioned to hit the incoming ball (right panel), he has time to aggressively place more shots down the line to the open court. We call attention to rallies in the supplemental video where players “work the point”, moving the opponent around the court.

In tennis, approaching the net is an important strategy, but requires the player to create favorable
Figure 7.4: Synthesized points feature emergent player behavior that is consistent with good tennis principles. (The red and blue heat maps visualize a KDE fit to the results of synthesized points, not the distribution used by our behavior models.) (a) When hitting the ball from left side of the court, virtual Nadal places most of his shot cross-court to Federer’s significantly weaker backhand side. When playing Djokovic, Nadal directs more shots down the line. (b) When virtual Djokovic has to move rapidly (to the right) to hit an incoming shot vs. Nadal, he plays defensively, placing most of his shots cross court even though that is where Nadal is standing. When Djokovic does not need to move quickly to reach the incoming ball, he plays more aggressively, placing a greater fraction of shots down the line to the open court. (c) Most of the time when virtual Federer approaches the net is after hitting a short incoming ball down the line to the open court. Positioning oneself at the net following a down the line shot is a common attacking strategy in tennis.

point conditions for the strategy to be successful. Figure 7.4-c visualizes Federer’s recovery position after hitting incoming cross-court shots from Djokovic. Notice that most of the time when Federer approaches the net is in situations where the incoming shot bounces in the front regions of the court (incoming shots that land in these regions are easier to hit aggressively) and when Federer places his shot down the line (allowing him to strategically position himself to cover his opponent’s return shot). Approaching the net after hitting a shot down the line (but not after a cross-court shot) is well known tennis strategy.

Overall, we believe our system synthesizes tennis animations that are substantially more realistic than prior data-driven methods for tennis player synthesis [20, 37]. Most notably, compared to recent work on GAN-based player synthesis [20], our characters make realistic tennis movements (rather than shuffle from side to side), and perform basic tennis actions, such as making contact with the incoming ball (they do not swing randomly at air) and recovering back into the court after contact.

7.3.2 User Study

To evaluate whether our player behavior models capture the tendencies of the real professional tennis players, we conducted a user study with five expert tennis players. All participants were current or former collegiate-level varsity tennis players with at least 10+ years of competitive playing experience. None of the study participants were involved in developing our system.
Figure 7.5: Participants rated our shot selection behavior model ($\mu = 3.84, \sigma = 0.83$) better than no behavior model ($\mu = 3.2, \sigma = 0.89$) but less realistic than the oracle ($\mu = 4.48, \sigma = 0.70$). We find all of these differences to be significant running Friedman’s non-parametric test ($p < 0.001, \chi^2 = 24.33$) followed by pairwise comparisons using Wilcoxon’s signed rank tests ($p < 0.019, Z = 44.50$ ours vs. no behavior, $p < 0.005, Z = 24.00$ ours vs. oracle). Similarly participants rated our recovery behavior model ($\mu = 3.76, \sigma = 0.76$) better than no behavior model ($\mu = 2.16, \sigma = 1.12$) but less realistic than the oracle ($\mu = 4.84, \sigma = 0.37$). Again we find all these differences to be significant running Friedman’s test ($p < 0.001, \chi^2 = 37.42$) followed by pairwise comparisons using Wilcoxon’s tests ($p < 0.001, Z = 19.50$ ours vs. no behavior, $p < 0.001, Z = 0.00$ ours vs. oracle).

We asked each participant to watch videos of points between Roger Federer, Rafael Nadal, and Novak Djokovic, generated by our system and answer questions (on a 5-point Likert scale) about whether a virtual player’s shot selection and recovery point behaviors were representative of that player’s real-life play against the given opponent. Note that the synthesized videos are generated using the video-texture based rendering approach introduced in the next Chapter 8. Each participant viewed 15 videos, five videos for each of three conditions:

**No behavior model.** This configuration generates the points without using our behavioral model. Specifically, we remove the shot match and recovery phase correction terms, ($C_{\text{shot}}$ and $C_{\text{recover}}$) from the cost computation during motion clip search. The only behavior constraint in this case is that clips chosen by our system must make ball contact.

**Our behavior model.** This configuration sets behavior targets using our full behavior model as described in Section 7.2. The behavior model is specialized to each matchup so that the distributions estimated for player behaviors are based on the data we have for the two players involved in the point.

**Oracle.** This configuration sets behavior targets to locations observed in real points from broadcast videos. In other words, we use our system to reconstruct a point from our database, directly using the database clips of the point with their original shot selection and recovery position, rather than
querying our behavior model.

Participants were not told which condition they were seeing and as shown in Figure 7.5 participants rated the behaviors of the players generated using our full behavior model as significantly more realistic (e.g. more consistent with their expectations of what the real-life player would do) than the baseline configuration using no behavior model. The difference was particularly pronounced when assessing recovery position ($\mu = 3.76$ for ours vs. $\mu = 2.16$ for no behavior model), where real-life players have well-known tendencies for where they recover. The difference was less pronounced for shot selection ($\mu = 3.84$ for ours vs. $\mu = 3.2$ for no behavior model), likely because in real play, players tend to vary their shot selection decisions (to keep an opponent off balance) making it more challenging for participants to confidently judge any one specific shot selection decision as atypical.

Finally, participants thought that the points reconstructed by our oracle model generated the most realistic behaviors, suggesting that while our behavior model is significantly better than the baseline model there are still aspects of player behavior that it does not yet capture. Nevertheless, we note that many of the participants also verbally told the experimenters that the task was difficult, especially when considering the videos generated using our behavior model or the oracle model. They often watched these videos multiple times carefully looking for behavioral artifacts.
Chapter 8

Synthesizing Photo-Realistic Appearances

In this chapter, we aim to address the last problem of synthesizing high-quality tennis animations – making them look photo-realistic. The classic graphics rendering pipeline requires sophisticated modeling of geometry modelling, rigging, skinning, as well as physics modeling of material, texture and lighting, which makes photo-realistic rendering of the synthesized motions a challenging problem. On the other hand, the large-scale broadcast videos contain abundant information about how players look in different motions. In this chapter, we combine the generated motions with image-based rendering techniques to construct photo-realistic video sprites that behave, move and appear like professional tennis players.

The major challenge lies in preparing a large database of real-world, single-viewpoint broadcast video clips for use in a video sprite rendering system. This involves use of neural image-to-image transfer methods to eliminate differences in player appearance over different match days and times, hallucinating missing pixel data when players are partially cropped in the frame.

In this chapter, we show that video textures [72, 73] can be integrated into the non-parametric tennis controller introduced in Section 4.3 to form an image-based animation system for synthesizing novel tennis videos. This system entangles the motion and appearance by reusing the original pixels from the source videos to maintain the realism of the synthesized animations. We demonstrate that we can synthesize videos of novel points that did not occur in real life, but appear as plausible points between star athletes on Wimbledon’s Centre Court and US Open Court (Figure 8.1). This also enables new experiences such as creating points between players that have never competed in real life (such as Roger Federer and Serena Williams), or interactive experiences where a user can control Roger Federer as he competes against a rival in the Wimbledon final.

Additionally, we show that motion and appearance can also be decoupled by leveraging the
CHAPTER 8. SYNTHESIZING PHOTO-REALISTIC APPEARANCES

Figure 8.1: Synthesized tennis videos of novel points. (a) Federer (near) vs. Federer (far) (b) Williams (near) vs. Federer (far) (c) Nadal (near) vs. Federer (far) (d) Williams (near) vs. Gauff (far). Note none of the match-up has ever happened in real life.

recent progress in neural appearance transfer techniques. The decoupling makes it possible to render unseen motions from the source videos, such as the physically simulated motions from Chapter 5 and Chapter 6. We demonstrate that the simulated motions can be synthesized into different tournament backgrounds.

8.1 Related Work

Controllable video textures. We adopt an image-based rendering approach that assembles frames for a novel animation from a database of broadcast video clips[16, 19, 72, 73]. Most prior work on generating controllable video sprites focuses on finding clip transitions where the local frame-to-frame motion and appearance are similar. While results are plausible frame-to-frame, higher-level behavior of the character can be unrealistic. Our player behavior models provide higher level control inputs that yield player sprites that behave realistically over the span of entire points.

The Tennis Real Play system[37] also creates controllable video sprite based tennis characters from broadcast tennis videos and structures synthesis using a simple move-hit-standby state machine that is similar to our shot cycle state machine described in Section 2.1 (they also use match play
statistics to guide shot selection to a limited extent). However, this prior work does not generate realistic player motion or behavior (see supplementary video for example output from Tennis Real Play). We utilize a video database that is nearly two orders of magnitude larger, a shot cycle state machine that differentiates pre-shot and post-shot movement, and more complex player behavior models to generate output that appears like the players as seen in broadcast video footage.

**Conditional Neural Models.** Vid2Game [20] utilizes end-to-end learning to create models that animate a tennis player sprite in response to motion control inputs from the user. However Vid2Game does not generate video sprites that move or play tennis in a realistic manner: player swings do not contact an incoming ball and player movement is jerky (see supplemental video for examples). While end-to-end learning of how to play tennis is a compelling goal, the quality of our results serves as an example of the benefits of injecting domain knowledge (in the form of the shot cycle structure and a data-driven behavioral model) into learned models.

**Image-to-image transfer.** Our system utilizes both paired [30, 88] and unpaired [111] neural image-to-image transfer to normalize lighting conditions and player appearance (e.g., different clothing) across a diverse set of real-world video clips. Many recent uses of neural image transfer in computer graphics are paired transfer scenarios that enhance the photorealism of synthetic renderings [3, 8, 32, 50, 87]. We use this technique for hallucinating occluded regions of our video sprites. Like prior work transforming realistic images from one visual domain to another [9, 10], our system uses unpaired transfer for normalizing appearance differences in lighting conditions and clothing in our video sprites.

### 8.2 Video Sprite based Animations

Given the court space position \( (x_p(t) \) from Equation 4.3), of a player and the video clip selected to depict the player during a shot cycle, we render the player by compositing sprites (players with rackets) from the source clip onto the court background image (a frame chosen from one of the broadcast video) with generated alpha mattes. Achieving high output quality required overcoming several challenges that arise when using broadcast video as input: maintaining smooth animation in the presence of imperfect (noisy) machine annotations, hallucinating missing pixels when player sprites are partially cropped in the frame, and eliminating variation in player appearance across source clips.

#### 8.2.1 Player Sprite Rendering

**Sprite Transformation.** We translate and scale the source clip sprite to emulate perspective projection onto the court (Figure 8.2). Given a source clip \( i \) with player bounding box (bottom
Source database clip Composition onto court (after translation/scaling)

Figure 8.2: Source video sprites are translated and scaled to render the player at a novel location in court space. Scaling requires an estimate of the court space extent of the player’s bounding box, which is computed from projection of the boxes’ bottom-left and bottom-right screen space points ($\tilde{x}_A$, $\tilde{x}_B$).

center) located at root position $\tilde{x}_p(t)$ (notations with $\tilde{\cdot}$ refer to positions in screen space), we compute the screen space translation, $\tau(t)$, of source pixels to the target court background image as:

$$\tau(t) = H^t x_p(t) - \tilde{x}_p(t),$$

where $H^t$ is the camera projection mapping the court space to screen space in the background image.

To determine the sprite’s scaling due to perspective projection, we use the bottom-left $\tilde{x}_A$ and bottom-right $\tilde{x}_B$ points of the source clip bounding box to compute its court space extent (using $H^t(t)$ for the source frame), translate this segment (in court space) so that it is centered about the player’s current position $x_p(t)$, then compute the screen space extent of the translated segment $||\tilde{x}_B' - \tilde{x}_A'||$. The ratio of the horizontal extent of the source clip segment and translated segment determines the required scaling, $\sigma(t)$, of the sprite.

$$\tilde{x}_A' = H^t \left( H^t(t)^{-1} \tilde{x}_A + (x_p(t) - x'_p(t)) \right)$$
$$\tilde{x}_B' = H^t \left( H^t(t)^{-1} \tilde{x}_B + (x_p(t) - x'_p(t)) \right)$$
$$\sigma(t) = \frac{||\tilde{x}_B' - \tilde{x}_A'||}{||\tilde{x}_B - \tilde{x}_A'||}.$$

Due to noise in estimates of a player’s root position $\tilde{x}_p(t)$, performing the above translation and
scaling calculations each frame yields objectionable frame-to-frame jitter in rendered output. To improve smoothness, we compute $\tau(t)$ and $\sigma(t)$ only at the start of the reaction phase, ball contact time, and the end of recovery phase, and interpolate these values to estimate sprite translation and scaling for other frames in the shot cycle.

**Clip Transitions** To reduce artifacts due to discontinuities in pose at clip transitions, we follow prior work on video textures [19] and warp the first five frames of the new shot cycle clip towards the last frame of the prior clip using moving least squares [71]. Linear interpolation of pose keypoints from the source and destination frames are used as control points to guide the image warping. We produce final frames by blending the warped frames of the new clip with the last frame of the prior clip.

**Matte Generation** To generate an alpha matte for compositing sprites into the background image of the tennis court, we use Mask R-CNN [22] to produce a binary segmentation of pixels containing the player and the racket in the source clip frame, then erode and dilate [6] the binary segmentation mask to create a trimap. We then apply Deep Image Matting [97] to the original frame and the trimap to generate a final alpha matte for the player. Matte generation is performed offline as an preprocessing step, so it incurs no runtime cost. In rare cases, we manually correct the most egregious segmentation errors made by this automatic process.

**Player Shadows** We generate player shadows using a purely image-based approach that warps the player’s alpha matte to approximate projection on the ground plane from a user-specified lighting direction. The renderer darkens pixels of the background image that lie within the warped matte. (See supplemental for a description of the control points used for this image warp.) This approach yields plausible shadows for the lighting conditions in our chosen background images, and avoids the challenge of robust 3D player geometry estimation (see Appendix A for more details).

### 8.2.2 Appearance Normalization and Body Completion

Using unmodified pixels from database player sprites results in unacceptable output quality due to appearance differences across clips and missing pixel data. To reduce these artifacts, we perform two additional preprocessing steps that manipulate the appearance of source video clips to make them suitable for use in our clip-based renderer. Both preprocessing steps modify the pixel contents of video clips used by the renderer. They incur no additional run-time performance cost.

**Hallucinating Cropped Regions** When the near court player in the source video is positioned far behind the baseline, part of their lower body may fall outside the video frame (Figure 8.3, top-left). In fact, 41%, 39%, 15%, 9% of the near court clips for Nadal, Djokovic, Federer and
Figure 8.3: When the player in an input clip lies partially outside the frame, we hallucinate missing pose keypoints from similar (fully visible) sprites, then use paired image translation to hallucinate missing pixels from the completed skeleton. In this example, the final sprite contains a combination of original pixels and hallucinated legs.

Serena Williams suffer from player cropping. To use these cropped clips in other court locations we must hallucinate the missing player body pixels, otherwise rendering will exhibit visual artifacts like missing player feet. We identify clips where player cropping occurs using the keypoint confidence scores from pose detection for the near court player and classify clips by the amount of body cropping that occurs: no cropping, missing feet, and substantial cropping (large parts of the legs or more are missing). We discard all clips with substantial cropping, and hallucinate missing pixels for clips with missing feet using the process below.

For each frame containing cropped feet, we find an uncropped frame of the same player in the database that features the most similar pose (Figure 8.3, top-center). We use the average L2 distance of visible pose keypoints as the pose distance metric. We transfer ankle keypoints from the uncropped player’s pose to that of the cropped player. (In practice we find the similar pose match in \(N=5\) consecutive frames from the uncropped clip for temporal stability.) Once the positions of the missing keypoints are known, we follow the method of [8] by rendering a stick skeleton figure and using paired image-to-image translation to hallucinate a player-specific image of the missing body parts (Figure 8.3, bottom). The matte for the hallucinated part of the player is determined by the background (black pixels) in the hallucinated output. We construct the final uncropped sprite by compositing the pixels from the original image with the hallucinated pixels of legs and feet.
CHAPTER 8. SYNTHESIZING PHOTO-REALISTIC APPEARANCES

Transformed source images (clothing matches target)

Source: images of Williams on day 2 (different clothing)
Target: images of Williams on day 1

Transformed source images (lighting matches target)

Source: images under different lighting conditions
Target: images from short time window (same lighting conditions)

Figure 8.4: We train CycleGAN models [111] to perform unpaired image-to-image translation to reduce differences in a player’s appearance across video clips. These corrections enable a larger number of video clips to be used as inputs for video sprite synthesis. Left: removing lighting differences. Right: modifying Serena Williams’ clothing (adding long sleeves) to match her outfit on a prior day.

Normalizing Player Appearance  Our database contains video clips drawn from different days and different times of day, so a player’s appearance can exhibit significant differences across clips. For example, the direction of sunlight changes over time (Figure 8.4-left), the player can fall under shadows cast by the stadium, and a player may wear different clothing on different days. (Serena Williams wears both a short sleeved and a long sleeved top, Figure 8.4-right.) As a result, transitions between clips can yield jarring appearance discontinuities.

Eliminating appearance differences across clips using paired image-to-image translation (with correspondence established through a pose-based intermediate representation [8, 32]) was unsuccessful because pose-centric intermediate representations fail to capture dynamic aspects of a player’s appearance, such as wrinkles in clothing or flowing hair. (Paired image-to-image translation was sufficient to transfer appearance for the players feet, as described in the section above, because appearance is highly correlated to skeleton in these body regions.)

To improve visual quality, we use unpaired image-to-image translation for appearance normalization (Figure 8.4). We designate player crops from a window of time in one tennis match as a target distribution (320 x 320 pixel crop surrounding the center point of the player’s bounding box), then train CycleGAN [111] to translate other player crops into this domain. To correct for lighting changes, we train a CycleGAN for each of our source videos to translate player crops from different times of the day to a distribution obtained from the first hour of the 2019 Federer-Nadal Wimbledon
Table 8.1: Offline preprocessing times (both per frame, and for all frames (total), on a TITAN V GPU) for operations needed to preprocess database video clips to prepare them for rendering.

<table>
<thead>
<tr>
<th>Task</th>
<th>Model</th>
<th>Total (hr)</th>
<th>Per frame (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segmentation</td>
<td>Mask R-CNN [22]</td>
<td>572</td>
<td>11200</td>
</tr>
<tr>
<td>Matting</td>
<td>Deep Image Matting [97]</td>
<td>36</td>
<td>350</td>
</tr>
<tr>
<td>Body Completion</td>
<td>Pix2PixHD [88]</td>
<td>.05</td>
<td>10</td>
</tr>
<tr>
<td>Appearance Norm</td>
<td>CycleGAN [111]</td>
<td>35</td>
<td>270</td>
</tr>
</tbody>
</table>

semi-final, from which we also pick a frame without players as our court background to ensure lighting consistency. When distribution shift in player appearance is due to changes of clothing across matches, we train CycleGAN models to transfer player appearance between specific matches.

8.2.3 Results and Evaluation

Table 8.1 shows the offline preprocessing time needed to prepare clips for rendering. We give results on a single TITAN V GPU but use frame-parallel processing across many GPUs to reduce overall preprocessing latency. Our implementation uses publicly available models for Deep Image Matting [97], Pix2PixHD [88] for body completion and CycleGAN [111] for appearance normalization. (See Appendix A for details.)

Overall, we believe the video sprite based animation system synthesizes tennis animations that are substantially more realistic than prior data-driven methods for tennis player synthesis [20, 37]. Most notably, compared to recent work on GAN-based player synthesis [20], our players do not exhibit the twitching motions exhibited by their output, make realistic tennis movements (rather than shuffle from side to side), and perform basic tennis actions, such as making contact with the incoming ball (they do not swing randomly at air) and recovering back into the court after contact. We refer the reader to the supplementary video to compare the quality of our results against these prior methods.

In addition to synthesizing points that depict tendencies observed in real-life points between the two participating players, we also use our system to generate novel matchups that have not occurred in real life (Figure 8.1). For example, Figure 8.1-a,b depicts Roger Federer playing points against himself and against Serena Williams on Wimbledon’s Centre Court. Figure 8.1-c shows Nadal playing against Federer on US Open court, which magically has never happened between the two players. Figure 8.1-d illustrates a highly anticipated match-up between the tennis legend Serena Williams and the young rising star Coco Gauff.

Limitations Our system generates visually realistic video sprites that often resemble broadcast video of Wimbledon or US Open matches. However, visual artifacts do remain in generated output. For example, player mattes exhibit artifacts due to errors in player segmentation, in particular when the far-court player overlaps a linesperson on screen. (Players and linespersons both wear
white clothing.) Second, appearance normalization is imperfect, so transitions between clips with dramatically different lighting conditions may exhibit appearance discontinuities. Finally, 2D pose interpolation is insufficient to remove animation discontinuities when transitions occur between frames with large pose differences. While our system benefits from a viewer’s gaze being directed at the opposing player during clip transitions (they occur when the ball is on the other side of the court), generating more realistic transitions is an interesting area of further work.

8.3 Neural Appearance Transfer

Alternatively, we can decouple motion and appearance by leveraging the recent progress in neural appearance transfer techniques. The decoupling makes it possible to render unseen motions from the source videos, such as the physically simulated motions from Chapter 5 and Chapter 6.

8.3.1 Approach

We adopt the representative work of GAN-based neural appearance transfer model, Vid2Vid [87] for translating 2D skeletal motion into photo-realistic player sprites. However, unlike its standard application of synthesizing full video with a relative large foreground person/face and almost static background, we need to synthesize relatively small players and place them in the appropriate location of the frame where the local background is rapidly changing. Vid2Game [20] proposed to train the model to predict the foreground mask for compositing synthesized foreground object into the background, which lead to artifacts such as visible seam at image blending boundary or missing body parts due to incomplete foreground mask generation.

Instead, we propose a modification to the Vid2Vid pipeline, where we input both the skeleton figure and the desired background image to encourage the model to focus on the synthesis of the foreground object while simply copying the background pixels in the region not occluded by the foreground object (Figure 8.5). We show that this modification can lead to more smooth and natural
8.3.2 Results and Discussion

Although it requires training a new Vid2Vid model for each player on one side from one court, once trained, they can be used to render the same motion into different tournament backgrounds. Figure 8.6 shows the sampled frames of such results, where we first render the simulated 3D motion as 2D skeletal figure on desired court background and then use the trained Vid2Vid models to synthesize photorealistic appearances.
Chapter 9

Conclusions

In this dissertation, I have shown that it is possible to use large-scale observations of athlete performance obtained from real-world video collections to create controllable, high-quality tennis animations of virtual characters playing singles tennis points. In particular, my solutions rely on both demonstrations acquired from video and physically-based simulations of tennis to achieve high-quality and controllable content generation. The demonstrations define the possible “space” of motions that can be made by the virtual character (e.g., via a low dimensional embedding of tennis motion), which significantly reduces the challenge of exploring how to move to perform a tennis task (e.g., simplifies control policy learning via RL). Demonstrations alone are not enough. Simulation is necessary to compute the outcomes that provide the feedback needed to learn control policies that generalize to novel situations. It would be difficult to address the challenging problem of high-quality and controllable content generation without either demonstrations or simulation. In the following sections I outline the broader implications of demonstration, simulation and the traditional graphics models for content generation.

9.1 Toward Foundation Models for Human Motion

Building motion foundation models from large-scale demonstrations has recently attracted lots of attention, similar to the domain of texts and images. The motion embedding model that I used in my hierarchical controller, as detailed in Chapter 5 can be viewed as a prototype for a motion foundational model, particularly tailored to tennis motions. It is used for motion generation in my work to constrain the space of possible tennis motions for the high-level policy to explore. Although not actioned in my work, it would be interesting to use the motion model as a prior for motion estimation from videos. For example, a strong prior on the space of possible motions could make it easier to extract more accurate motion. Looking into the future, I believe the development of a general purpose motion foundation model will be the next milestone in motion modeling. Motion
foundation model can serve as the important data prior for improving motion estimation in challenging situations where occlusion or ambiguity poses challenge for vision-based motion estimation. Furthermore, a motion foundation model stands to enable wide applications of conditional motion generation as well as developing a universal tracking controller that can imitate a large variety of human motions. Most recently, we have seen promising work in building general motion foundation models, for applications such as text-to-motion [84]. It’s worth noting, however, that the ongoing efforts rely solely on motion data derived from publicly available motion capture datasets like AMASS [54]. These repositories, though valuable, pale in comparison to the vast volume of text and image data available on the Internet. Therefore, I believe video will be the ultimate source of human motion data for building motion foundation models.

9.2 The Role of Simulation

Simulation plays an important role in my work to fuse the gap from existing demonstrations to novel situations. However, I would like to remind the readers that the simulation model used in my current work is relatively simple. The character is modeled as articulated rigid bodies actuated with ball joints. The racket head and the tennis ball are also modeled as rigid bodies, and the ground is modeled without considering real-world materials such as grass or clay. Despite the simplified modeling and simulation, we are excited to see impressive characters created from the current system. I am curious to see whether employing more sophisticated modeling would lead to more accurate simulation of real-world tennis play. For example, can we replace the joint actuated character model with anatomically detailed [41] or muscle actuated human model [31]? Can we simulate deformation of the racket strings and the ball during ball contact? Given these more accurate simulation, it would be interesting to see whether we can more accurately reproduce fine-grained motion details of tennis swings, such as the wrist pronation to increase spin and longer follow-through to slow down the racket. Furthermore, can we model longer-term aspects such as fatigue and play history into the simulation, and see whether the character can discover real-world strategies through extensive self-play, such as holding the racket with both hands to allow the primary hand to rest between shots. I believe more accurate simulation would be critical if we expand the application domain from tennis to broader areas where the interaction between human and the environment becomes more complex.

9.3 The Role of Traditional Graphics Models

The work in this thesis started with the goal of synthesizing novel tennis videos, which was carried out by the 2D sprite based animation system. During my exploration to further improve the result quality of the synthesized video via learning based approach, I realized that it is challenging to
develop an end-to-end solution. Instead, it makes the learning problem a lot more tractable if we adhere to the traditional graphics pipeline by first generating controllable motion and then addressing the photorealistic appearance synthesis. I believe this paradigm also holds true when pursuing high-quality and controllable content generation across a wider spectrum. For example, we have seen great success in recent foundation models for image synthesis [70], while the fine-grained controllability of these models still hinges on conditioning the generation on the sketches or edge maps which have been explored in the graphics field decades ago. Furthermore, video synthesis via foundation models such as text-to-video [4] still suffers from unnatural and discontinuous motion of the subjects in the synthesized video. I believe considerable effort should be spent on improving the condition on motion generation to enhance the quality of synthesized videos, which also echos the traditional graphics pipeline. Overall, in an era where the magnitude and diversity of data are increasingly emphasized, it is important not to underestimate the efficacy of traditional graphics models and pipeline abstractions. I am confident that by integrating the wealth of knowledge the field has about graphics representations into modern data-driven techniques, we would see more powerful systems constructed for high-quality and controllable content generation.
Appendix A

Implementation Details and Additional Results

Hierarchical Motion Controller

Low-level Imitation Policy

Network The policy is modeled by a neural network that maps a state \( s \) to a Gaussian distribution over actions \( \pi(a|s) \) with an input-dependent mean \( \mu(s) \) and a fixed diagonal covariance matrix \( \Sigma_\pi \). The mean is specified by a fully connected network with 3 hidden layers of \([1024, 1024, 512]\) units, followed by linear output units. The value function \( V(s) \) is modeled by a similar network, but with a single linear output unit. All the hidden units use ReLU activations [59].

Rewards In all the experiments, we manually specify the weights and scales as follows: \( \omega_o = 0.6, \; \omega_v = 0.1, \; \omega_p = 0.2, \; \omega_k = 0.1, \; \omega_e = 0.01, \; \alpha_o = 60, \; \alpha_v = 0.2, \; \alpha_p = 100, \; \alpha_k = 40 \).

Training The low-level policy is trained with 8,192 environments with a simulation frequency of 120 Hz. Hyper-parameters used during training of the low-level policy is available in Table A.1. The low-level policy is first trained using AMASS dataset with about 1 billion samples. Next, the low-level policy is fine-tuned using the kinematic motion dataset \( M_{\text{kin}} \) we extracted from tennis videos with about 1 billion samples, which can then be used to correct the kinematic motions and create the physically corrected motion dataset \( M_{\text{corr}} \). The low-level policy used in the control hierarchy for controlling the character’s low-level movements is further fine-tuned using \( M_{\text{corr}} \) for each different player with about 0.25 billion samples.

To further remove the reliance of residual force control, we can keep fine-tune the policy by gradually reduce the maximum allowed residual forces/torques with about 1 billion samples, while
TABLE A.1: Hyper-parameters for training the low-level policy

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation frequency (Hz)</td>
<td>120</td>
</tr>
<tr>
<td>$\Sigma_\pi$ Action distribution variance</td>
<td>0.03</td>
</tr>
<tr>
<td>Samples per update iteration</td>
<td>262144</td>
</tr>
<tr>
<td>Policy/value function minibatch Size</td>
<td>16384</td>
</tr>
<tr>
<td>$\gamma$ Discount</td>
<td>0.99</td>
</tr>
<tr>
<td>Adam stepsize</td>
<td>0.00002</td>
</tr>
<tr>
<td>GAE($\lambda$)</td>
<td>0.95</td>
</tr>
<tr>
<td>TD($\lambda$)</td>
<td>0.95</td>
</tr>
<tr>
<td>PPO clip threshold</td>
<td>0.2</td>
</tr>
<tr>
<td>Episode length</td>
<td>300</td>
</tr>
</tbody>
</table>

the residual forces are reduced by 1% about every 10 million samples. We find it critical to slowly decrease the residual forces with sufficiently long training time.

**Motion Embedding**

**Network** The encoder is a three-layer feed-forward neural network, with 256 hidden units in each internal layer followed by ELU activations [13]. The output layer has two heads for $\mu$ and $\sigma$, required for the reparameterization trick used to train VAEs [34]. The decoder uses mixture-of-expert (MoE) architecture. Specifically, the MoE decoder consists of six identically structured expert networks and a single gating network to blend the weights of each expert to define the decoder network to be used at the current time step. Similar to the encoder, the gating network is also a three-layer feed-forward neural network with 256 hidden units followed by ELU activations. The input to the gating network is the latent variable $z$ and the current pose. Each expert network is also similar to the encoder network in structure. These compute the next pose from the latent variable $z$ and the current pose.

**Training** Hyper-parameters used during training of the motion embedding model is available in Table A.2. We adopt scheduled sampling [1], where a sample probability $p$ is defined for each epoch. The predicted pose is used as the input for the next time step with probability $1 - p$, otherwise, the ground-truth pose is used. The entire training process is divided into three modes: supervised learning ($p = 1$), scheduled sampling (decaying $p$), and autoregressive prediction ($p = 0$). The number of epochs for each mode is 50, 50, and 400 respectively. For the scheduled sampling mode, the sampling probability decays to zero in a linear fashion with each learning iteration.

To train the model for predicting the motion phase with limited supervision (only 20% of the data is labeled with motion phase), we adopt a curriculum similar to scheduled sampling. We define a sample probability $q$, which specifies the probability of sampling a motion sequence labeled with motion phase. The entire training process is also divided into two stages: $q$ decays linearly from 1 to 0.1 in the first stage, and stays at $q = 0.1$ for the second stage. Each stage is trained for 250
Table A.2: Hyper-parameters for training the motion embedding model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latent space dimension</td>
<td>32</td>
</tr>
<tr>
<td>Number of frames for condition</td>
<td>1</td>
</tr>
<tr>
<td>Number of frames for prediction</td>
<td>1</td>
</tr>
<tr>
<td>Sequence length</td>
<td>10</td>
</tr>
<tr>
<td>Number of seqs per epoch</td>
<td>50000</td>
</tr>
<tr>
<td>Batch size</td>
<td>100</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

epochs.

**High-level motion planning policy**

**Network** We adopt the same network architecture as the low-level policy.

**Ball trajectory prediction model** The ball trajectory prediction model is used for estimating future incoming ball trajectory as the observation for the high-level policy, as well as estimating the outgoing ball bounce position for computing the ball position reward. At the offline stage, we compute a large ball trajectory pool by densely sampling the plausible ball states at launch time, including the height of the ball, the velocity of the ball and the spin velocity of the ball. The sample steps we used are 0.1m, 0.1m/s and 0.5 RPS, respectively. To reduce complexity, all the trajectories are computed in the Y-Z plane. The computed trajectories are stored in a dense matrix used as a lookup table. At runtime, a particular trajectory can be estimated by indexing the lookup table with the ball’s launch state, rounded by the sample steps we used. In practice, we find this ball trajectory prediction model provides efficient and accurate estimates of the future ball positions and bounce position.

**Rewards** In all the experiments, we manually specify the scales as follows: $\alpha_r = 0.05$, $\alpha_\phi = 0.1$.

**Training** The high-level policy is trained with 30,720 environments. Hyper-parameters used during training of the high-level policy (with our proposed curriculum of three stages) is available in Table A.3.
Table A.3: Hyper-parameters for training the high-level policy with the curriculum of three stages.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation frequency (Hz)</td>
<td>120</td>
<td>360</td>
<td>360</td>
</tr>
<tr>
<td>$\Sigma_\pi$ Action distribution variance</td>
<td>0.25</td>
<td>0.04</td>
<td>0.0025</td>
</tr>
<tr>
<td>Samples per update iteration</td>
<td>327680</td>
<td>983040</td>
<td>983040</td>
</tr>
<tr>
<td>Policy/value function minibatch Size</td>
<td>16384</td>
<td>16384</td>
<td>16384</td>
</tr>
<tr>
<td>$\gamma$ Discount</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>Adam stepsize</td>
<td>0.0001</td>
<td>0.00002</td>
<td>0.00001</td>
</tr>
<tr>
<td>GAE($\lambda$)</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>TD($\lambda$)</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>PPO clip threshold</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Episode length</td>
<td>600</td>
<td>300</td>
<td>300</td>
</tr>
</tbody>
</table>

Behavior Modeling

Point State Descriptor Computation

In our behavior model, we build a point state descriptor for a point’s current state by discretizing the features which influence player’s behavior decisions. The details of the discretization for each feature can be found in Table A.4. We divide each side of the court into six regions. Horizontally, we split the court into three regions (Ad/Center/Deuce). The three regions are equally divided within the side boundaries for singles and the non-center regions are extended to their open areas. Vertically, we split the court into Front and Back given the middle line between service line and baseline. This simple discretization can capture player and ball characteristics such as cross-court vs. down-the-line shot and ground strokes vs. net plays. The origin of our court coordinate system lies at the court’s center, with a horizontal range spanning $\pm 4.12$ m and a vertical range spanning $\pm 11.89$ m. When we compute the descriptor for the player on the far side, we rotate the court by 180 degrees to make them consistent.

Table A.4: Discretization for each feature in the point descriptor. For the horizontal bins, “Ad/Center/Deuce” are split at $x = -1.4$ m and $x = 1.4$ m. For the vertical bins, “Front/Back” are split at $y = 9$ m. For the velocity to reach the contact point, we discretize the continuous velocity into five bins by 1.2 m/s, 1.8 m/s, 2.4 m/s and 3.0 m/s.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Horizontal</th>
<th>Vertical</th>
<th>Bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player’s position at contact</td>
<td>Ad/Center/Deuce</td>
<td>Front/Back</td>
<td>6</td>
</tr>
<tr>
<td>Opponent’s position at contact</td>
<td>Ad/Center/Deuce</td>
<td>Front/Back</td>
<td>6</td>
</tr>
<tr>
<td>Incoming ball’s start position</td>
<td>Ad/Center/Deuce</td>
<td>N/A</td>
<td>3</td>
</tr>
<tr>
<td>Incoming ball’s bounce position</td>
<td>N/A</td>
<td>Front/Back</td>
<td>2</td>
</tr>
<tr>
<td>Velocity to reach the contact point</td>
<td>N/A</td>
<td>N/A</td>
<td>5</td>
</tr>
<tr>
<td>Shot placement position</td>
<td>Ad/Center/Deuce</td>
<td>N/A</td>
<td>3</td>
</tr>
</tbody>
</table>
Extended Evaluation

Figure A.1: Shot placement (forehand/backhand) and recovery positions of Federer (right-handed), Djokovic (right-handed), and Nadal (left-handed) in our simulated points (top) and database points (bottom). By modeling behaviors from real-world data, we can simulate points that capture player-specific styles that are consistent with well-known tendencies of the real-life players.

Recreation of player style  Fig. A.1 compares the shot placement and recovery position of all three male players in the in the real world data from our database with the results of simulated points generated using our behavior models. Note that in general, our simulated points share similar statistics in important player behavioral decisions, including shot selection (shot type, shot velocity and shot placement) as well as recovery positions, which are explicitly modeled by our behavior models.
Appearance Synthesis

![View of player shadow](image)

Figure A.2: Image-based approach for player shadow rendering. We warp the player’s alpha matte to approximate projection on the ground plane from a user-specified lighting direction. We warp the matte ten times around the given lighting direction to simulate soft shadow (d) and darken the pixels in the warped matte to render the final player shadow.

Player Shadows

We generate player shadows using a purely image-based approach that warps the player’s alpha matte to approximate projection on the ground plane from a user-specified lighting direction (as shown in Fig. A.2). Specifically, we choose three points to control the image warping: two ankles points and one head point (Fig. A.2a). To obtain the positions of these control points in the target image, we fix the two ankle points and rotate the head point around the middle of the two ankle points given a user-specified angle, then use moving least squares [71] to warp the player’s alpha matte to compute the shadow region (Fig. A.2c). We simulate the effect of soft shadows by warping the matte ten times around the given angle (Fig. A.2d). Finally, the renderer darkens pixels of the background image that lie within the warped matte to render the player shadow (Fig. A.2b). This simple approach yields plausible shadows for the lighting conditions in our chosen background images, and avoids the challenge of robust 3D player geometry estimation to perform physically correct shadow rendering or leveraging learning based method to train a generative model for shadow synthesis.
Appearance Normalization and Body Completion

For appearance normalization, we directly use the Cycle-GAN implementation from [111]. We trained 12 models for both player relighting and re-clothing tasks from scratch, including: 10 relighting models for the five broadcast videos in our database (each model for near/far side of the court) and two re-clothing models for Serena Williams to modify her clothing from one video to look like her clothing from the other (plus each model for near/far side of the court). To train each model, we collect 10K images from each visual domain. Training each model takes approximately 24 hours to converge using the default training parameters from [111] on a TITAN V GPU.

For body completion, we use the Pix2PixHD implementation from [88] to convert a stick figure rendering of a players skeleton to a realistic sprite. We trained four models from scratch, for each of the players standing on the near side of the court. To train each model, we collect 5K image pairs. The training takes approximately 12 hours to converge using the default parameters of the code on a TITAN V GPU.

Extended results

Figure A.3: Montage of appearance normalization results. The top row shows the montage of uniformly sampled cropped frames around nearside players throughout each source video (ordered by day time). The bottom row shows the montage of the corresponding frames with appearance normalization, which are processed to share the same lighting conditions.

Fig. A.3 illustrates the overall performance of our appearance normalization for relighting the tennis players into uniform lighting condition. For each source video, we uniformly sample the nearside players throughout different times of the match and make a montage of the cropped frames around the player. We visualize the montages before and after normalization in the top and bottom rows. Despite the dramatic lighting variance throughout the first two matches and the overall
brightness/saturation difference for the other three matches, our appearance normalization is capable of bringing these diverse lighting distributions back to the same lighting condition.

Fig. A.4 shows randomly sampled results of relighting the three male players, as well as both relighting and re-clothing Serena Williams.

Fig. A.5 shows the randomly sampled results of our body completion approach for Nadal and Djokovic, who have more cropped frames as they usually stand further back compared to the other players. Our approach successfully complete most of the cropped bodies.
Figure A.4: Appearance normalization results for relighting the male players (top) and relighting and re-clothing Serena Williams (bottom). The normalized sprite is shown in the black box and the original sprite is shown to its left. Notice how normalization corrects for significant sunlight differences that occur over men’s matches that take place outdoors over the span of hours, and adds long sleeves to William’s shirt. (high-resolution image—please zoom in)
Figure A.5: Body completion results for Nadal (top) and Djokovic (bottom). The completed sprite is shown in the black background and the original cropped sprite is shown to its left. (high-resolution image—please zoom in)
Bibliography


[68] Paul Power, Hector Ruiz, Xinyu Wei, and Patrick Lucey. Not all passes are created equal: Objectively measuring the risk and reward of passes in soccer from tracking data. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, New York, NY, USA, 2017. Association for Computing Machinery.


