Intuitive Physics of Bouncing Spheroids

https://github.com/miniyoung84/balls

Anders Vaage vaage010 | April Roszkowski roszk008 | Desmond Kamas kamas002 | Chase Choi choix698

Abstract

Humans have a strong intuitive sense of the trajectory of a ball as it makes contact with another plane by inferring physical properties of the ball[11]. In order to emulate this effect in both simulated and real environments, we have created a Convolutional Neural Network (CNN) known as the Early Fusion Bounce Network (EFBNet) that takes in video data to predict the position and trajectory of ellipsoids. We created a generator to produce a diverse simulated dataset randomized across dozens of physical axes for training our CNN, and used this generator along with a manually created dataset of real-world data to test the model. The results show intuitive physics using EFBNet works exceptionally on simulated data. It also shows strong promise for positional predictions on real-world data with a comparable error to the baseline model PIM[20].

1. Introduction and Motivation

In the real world, an observer has limited information to understand the scene before them. Humans have been shown to have an ingrained sense of projectile motion, especially when presented with visual data[11]. This involves a neural processing of certain properties of the moving object such as its shape, material, restitution, etc. as it affects its current movement in the air.

Of course in a simulated environment, emulating this idea for an AI could be as simple as handing it the keys to the exact algorithm that creates the underlying factors of physical interactions. However, this would not be intuitive physics nor would it be an useful emulation of human opponents; whether they are simulators or game engines, in a simulated sense this would not be cutting-edge emulation. In a real world environment, there are different factors to consider. This could include a situation that forces vision-based analysis for data in a scenario that cannot be reproduced or re-analyzed and only has limited visual stimulus available. One area where such a situation may arise is the field of robotics. Since many of the simulated environments are generally used to emulate real-world physics, a working knowledge of intuitive real-world physics could be reinforced by a simulated one and vice versa. There is work that must be put in to achieving that which will be covered in this paper.

Our research question is as follows: Given a video of a elliptical object falling with the force of gravity and colliding with the ground plane, can we predict the trajectory and position of the ellipsoid based on the past trajectory? This is especially intriguing for cases where the trajectory vector appears to be random for shapes that are not perfectly spherical and take a more irregular shape.

Previous research typically covers a single aspect of this issue, but nothing comprehensive. One main component of existing research is the proof that neural networks can be used to a) predict outcomes using data in a manner similar to humans by learning from history to infer context rather than relying solely on the current context, and b) use this method to make predictions of physical properties and outcomes. Although these methodologies are close to the end goals for our experiments, they generally lack in the ability to create a diverse data set and thus test in very specific controlled environments or for very specific situations. This does not allow us to get a dynamic model for general or practical situations.

Our research provides a tool that can generate a diverse set of simulated physical interactions in a diverse environment that can auto-generate a database of practically any size for various training. We also provide a neural network that can visually train on samples in an .npy format that can learn how different objects behave in different environments. Lastly, we have manually created a data-set of real-world samples in order to verify its accuracy and its use outside of simulated environments.

2. Related Work Study

The 3D-PhysNet architecture can be applied for the object when another solid acts a force upon it; especially in a simulated environment. [25] Previous work has also been performed in how to properly manipulate these environments to gain more information for prospective studies. [19] Various advancements that have been made in this research

field for the purpose of video games which must also consider different materials and their physical properties in simulated environments. [24].

It is important to consider the underlying factors in the behavior of an object in a physical environment. Prior research approaches this with an approach similar to how humans think. In order to accurately evaluate how an object would move, one has to first calculate the underlying factors behind an object's movement. [3] These studies do exactly that in order to possibly locate the aspect that contributed to that movement. [26] There are papers that also have boiled down this philosophy into computational terms even when coming from a human-centered mindset, mostly through a stochastic approximation of a Bayesian inference by recalling collisions of multiple objects. [2] Work in this context has to also consider how objects interact with each other while keeping in mind how to classify what an active object is and what is individually but trivially contributing to the ambient environment. [4]

Models must be trained to test different data sets such as ones for collisions. [21] Some tangential research may have been able to provide a baseline on how to evaluate forces that are applied to a solid such as a poke from a uniformly controlled robotic hand [1]

Many other papers seem to be run on simulations that give underlying information to the user. This is the biggest difference between a simulated and real environment. One must consider different methods such as dealing with noise and cluttered environments. [9]

There are several machine learning methods which would assist in combining both simulated and real environment in a data-set may be proved useful not only for sample size, but also verifying the accuracy of any such simulation. [15] Prior research has proven that deep unsupervised convolutional networks can be used for computer vision based reinforcement learning methods. [13] In fact there are implementations that can save significant amounts of time while still providing strong results even on raw visual input data. [15]

Materials also come into play in prior research. For example the physics of blocks that are specifically made of wood have been implemented for a 3D simulation [16]. Once again that is backwards from our intentions of attempting to see what material something is made of (versus already knowing what it is made of). Other examples include billiard balls colliding with each other in a simulation which are made of polyester in perfect spheres [7], specific human interaction with objects [8], and the physics of rope that are generally made of strung up hemp. [17] [22]

A physical emulation using computer vision techniques can allow us to infer more physical properties of an environment and object with limited visual stimuli. We will apply engineering concepts with a similar physical work in order to significantly extend its reach and apply a practical benefit to it. [14]

A project that we should implement is one that can extract information from either raw visual data or a simulation. One paper focuses on predicting meaningful forces whose effects lead to accurate imitation of the motions observed from videos. [6] This marries two concepts that can be used in order to work between the two mediums. Similar work has been done in the field of intuitive physics, where a model predicts the next frame given an input video.Using one of these models, we could back-propagate the coefficient of restitution using the predicted frames. Another important thing to consider is the ability to process a large data-set once it's generated. Previous work exists using Convolutional Neural Networks with a massive video dataset. [12] Although this work revolves around a much simpler classification than we're hoping to reach, it is a strong baseline method that is proven on a dataset that is order of magnitude larger than what it is required for our intuitive physics model.

3. Baseline

For our baseline, we compare against a similar approach by Purushwalkam et al. 2019 which also predicted post-bounce locations and trajectories, although for arbitrary surfaces and only for perfect spheres[20]. While their full model aims to predict bounce trajectories given a single still image and the bounce location, we look specifically a component of their model, which they call the Physics Inference Module (PIM). The PIM takes as input ten frames preceding the bounce as well as physical parameters including the coefficient of restitution to predict the following ten frames of the sphere's trajectory. The median error of these predictions was approximately 10cm over a training set of simulated data, with the real world error for their Core Trained PIM ranging from around 15cm to 30cm depending on the estimated coefficient of restitution. Our algorithm is different in that it does not explicitly seek to encode the underlying physics of the trajectory and instead learn it implicitly from the training data. In addition, we predict only the three-dimensional velocity and position vector instantaneously after a bounce, not the frames after which might exacerbate any error.

4. Methodology

4.1. Dataset Generation

The tool we developed in order to generate a massive dataset of simulated videos of different spheroids bouncing off of the ground plane. Simulation was done using a Python module called PyBullet [5], and several scene factors are randomized. The first randomized factor is the textures that comprise the ground plane to simulate visual dif-

ferences in the environment. Other visual variations include the color of the spheroid which is randomized for the same reason via an (r,g,b) tuple parameter. A camera was also setup in a randomized three-dimensional coordinate for its location and another randomized three-dimensional vector to represent the angle at which it was viewing the spheroid. This created a strong sense of the kind of limited information an agent would have when viewing these situations (i.e. things that involve interacting with spheroids typically isn't from the same point of view).

A spheroid is created using a geometric mesh using various randomized ellipses, masses, and restitution in order to simulate spheroids made of different material, mass, size along with the levels of inflation and deformity during collisions. A randomized scale factor is also found and applied to each ellipsoid mesh in order to reduce the impact of scale discrepancies and misreadings during the domain transfer. Another three-dimensional vector is randomized in order to create a point for which the spheroid is released from its theoretical grip. Lastly, the lighting in the scene is randomized through two three-dimensional vectors: one for its location and another for its properties which include diffuse, ambient, and specular.

Once these "objects" in the environment are all set and scripted, its animation is activated, but our script only records crucial information. In order to standardize its output, files are stored in .npy format. These can be read in to process a visual format, but also just read in as pure data. It should be noted that every video is designed to be 30 frames with its first collision with the plane being in the middle (14/15th frame). This allows for a high degree of diversity in the samples while providing the right uniformity to train a neural network. Data is recorded until the second bounce which is compared with its first bounce to obtain its orientation and scale of movement. The final position, orientation, and velocity at the time of the second bounce is recorded specifically for each sample into a dictionary (hence the .npy file). This entire process is run on a loop to generate many samples. For each of our runs, we generated 30,000 samples for a training set along with 1,000 additional samples for a test set.

It must be noted that we also used concepts based around domain randomization to complete the bridge between simulated datasets and real world datasets. It was not practical for us to get a large enough sample in order to train on real world so a form of domain transfer was required. Due to the convenience that our data generation script provides, this was the optimal way to build our model. Domain Randomization involves the training on a large and diverse domain that could represent many different possible environmental setups. It is expected that a majority of these environments are not representative of the real-world. However EFBNet would train on all of these environments including the one that is the most representative of reality. When a real world sample is predicted, the model should be able to recognize and account for that environment. This is a well-established form of domain transfer. [23]

4.2. CNN: Early Fusion Bounce Network

We've adapted a version of the ImageNet architecture in order to implement the Early Fusion Bounce Convolutional Neural Network ("EFBNet") as proposed by Karpathy et al. [12]. The main change made to the ImageNet architecture was modifying the first set of image filters to convolve in 3 dimensions: the spatial dimensions of the clip (uv coordinates) and also the time dimension. Essentially, this allows the first convolution layer to combine information from all frames of the clip at once. In shorthand, the full architecture is $C(96, 11, 3, 10) - N - P - C(256, 5, 1) - N - P - C(384, 3, 1) - C(384, 3, 1) - C(256, 3, 1) - P - FC(4096) - FC(4096)$ where $C(d, f, s)$ indicates a convolutional layer with d filters of spatial size $f \times f$, applied to the input with stride s. The first layer is of the form $C(d, f, s, t)$, where t is the number of frames in each clip. $FC(n)$ is a fully connected layer with n nodes. P are spatial max-pooling layers of dimension 2×2 with a stride of 2, and N are ReLu normalization layers. The input of this network are tensors of shape (C, T, H, W), where C is the number of color channels, T the number of frames in the clip, H and W the dimensions of a single image. In our case, our tensors were of shape $(3, 10, 170, 170)$. The output of the network is a vector in $\mathbb{R}^6$, where the first 3 entries are the predicted landing position of the spheroid, and the last 3 are the predicted velocity of the spheroid after that landing. All positions are given relative to the camera, and its associated loss function is simply Euclidean distance from the ground truth $\mathbb{R}^6$ vector.

Initial results with this network were lacking, as the network accurately predicted the direction of where a spheroid lands after its first bounce, but had massive error in the magnitude of that position vector. It was suspected that this was due to the network having no conception of scale for each clip, so predicting the precise distance of a point relative to the camera was impossible. Our attempt to remedy this was to amend the input to EFBNet to also include the length of the spheroid's longest axis, which we call a scale factor. There is no change to the architecture, we simply add the scale factor to the flattened input vector of the first fully connected layer.

EFBNet is implemented using a python library known as PyTorch. [18] Once the training begins, our implementation records the loss along with the most recent model. This is implemented with a 5 layer CNN. Additionally, in order to account for the scale of each sample, a scale label is found when generating data and is applied to every sample in the CNN. This was to allow for easy manual runs with the al-

gorithm that could be stopped without the loss of progress easily or the resuming of training if it were to be interrupted. Previous research indicates that this process would be very slow and command a high-cost for training [10]. For this reason, it was trained using machines at The Minnesota Supercomputing Institute (MSI) on machines with powerful GPUs in order to take advantage of NVIDIA CUDA.

Refer to our repository for the complete codebase[1].

4.3. Real World Data Collection

We manually created 63 real world clips to test on, using 13 different spheroids. Videos were captured of spheroid's being tossed/dropped in front of the camera, such that the first bounce's location was in frame. These videos were manually edited down to 10 clips, cropped to a center square, then resized to 170×170 pixels. Due to our limited tools during the COVID-19 pandemic, we only had single-camera setups and, as such, were unable to get precise measurements on true velocity for real-world samples. The ball's second bounce location was accurately recorded, though. We also measured the longest axis of each spheroid, as additional input to EFBNet. Ideally with better equipment, we would've been able to also record accurate velocities for real-world samples.

4.4. Evaluation

In order to test our model, the latest model predicted position and velocity for each video. For values where we could not properly measure velocity, we used a placeholder value which is our results analysis does not include real-world velocity estimation. Future work would include analysis of the model's accuracy in prediction of real-world post-bounce velocities. Euclidean distance, cosine distance, and percent error in vector magnitude are used as our metrics to evaluate our quantitative results.

5. Results

Our model was tested on simulated dataset of 1000 generated samples.

The median error in position was 0 over all samples. This means that the predicted position using a converged model from EFBNet was successfully able to predict the trajectory of almost every single sample. Figure 1 shows the same effect is seen when measuring with cosine distance on the simulated test dataset with a zero median error. It should be noted that the possibility of over-fitting was addressed by verifying the models after various points of convergence. What we found was that the differences in the results were trivial when considering the change in error with each epoch. Considering this fact and the size of the

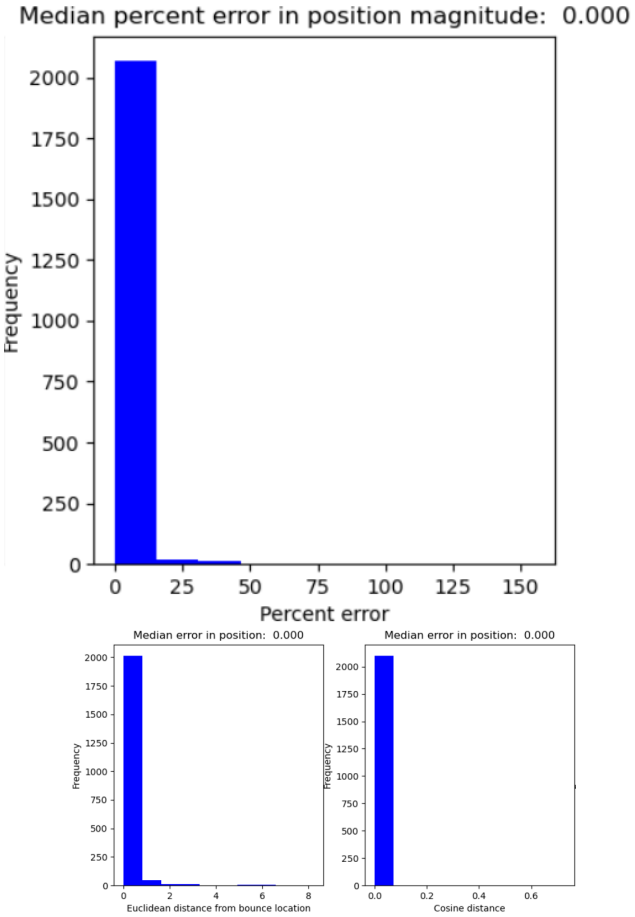

Figure 1. The percent error in the magnitude of position, above, and the Euclidean and cosine distances, below, based on 1000 unseen simulated videos.

sample, we can say with confidence this is a proven model for consistent simulated samples.

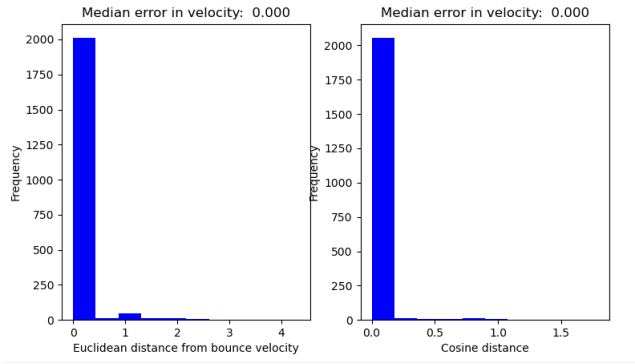

Figure 2. The median errors in velocity over a set of 1000 unseen simulated videos.

We can see the performance of the model on the final velocities of spheroids as seen in Figure 2 for Euclidean distance which was low, indicating high accuracy. Figure

[1] https://github.com/miniyoung84/balls

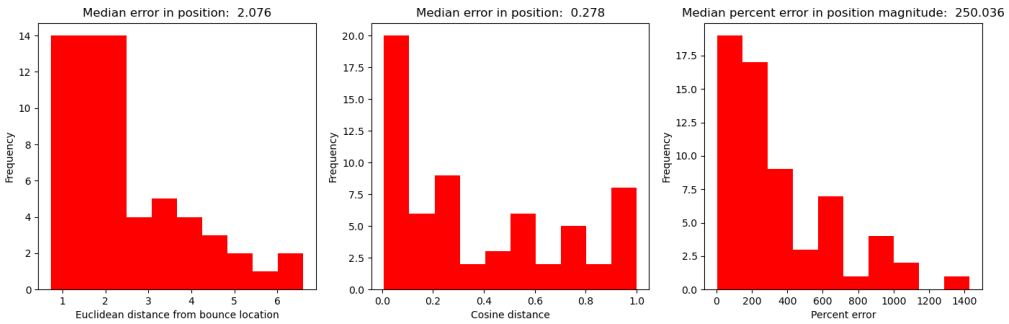

Figure 3. Results of real-world testing using a set of 63 manually collected videos.

2 shows a similar result with cosine distance with a low error. This proves our model can also accurately predict the direction of the final velocity of simulated data.

We were able to test on the real world dataset to mixed results (see Figure 3). Ideally, we would have more real-world samples to evaluate with, but due to time constraints, we only collected 63 samples. As such, we advise the reader not to take these results as fact: it is highly possible that our samples were recorded under ideal (or sub-optimal) conditions for EFBNet's classification. For example the real world data sample featured many different spheroids, initial angular velocities, and initial velocities, on top of having different backgrounds, lighting conditions, etc.

Even after incorporating scale into EFBNet via the longest axis measurement, our model struggles to predict the magnitude of the position vector, more so than PIM. That said, the cosine distance for the direction of the trajectory is generally low (median of 0.278) which indicates promise about EFBNet's ability to predict the direction of the spheroid's bounce, if not the distance. More investigation is required in order to verify potential problems and promise areas.

6. Conclusion

6.1. Limitations and Future Work

Extensions of this work would see use of proper equipment to accurately measure velocities of real-world spheroids. This would allow us to properly investigate the data with the direction of the trajectory vector of the test set and the implications of this method of domain transfer. Other possible extensions include use of a more robust domain transfer method, and higher-fidelity simulation of training data. Our survey of possible domain transfer methods showed that another strong possibility for domain transfer would be training a separate Generative Adversarial Neural Network (GAN) to transform real-world samples into the simulated domain, or vice versa. For simulation, PyBullet which was easy to set up and use, but doesn't offer the fidelity of dedicated rendering software like Blender or Maya. Future work may also attempt to generate simulated data which appears closer to our reality, to ease the process of domain transfer.

6.2. Summary

Using the CNN EFBNet, we were able to successfully predict trajectories and velocities of objects in a simulated environment. Although our algorithm models the position and velocity of a bouncing spheroid, as opposed to the actual motion profile by the baseline PIM algorithm, given accurate position and velocity estimations we can solve for the full profile using basic Newtonian physics. That said, EFBNet seems to perform comparably to the PIM model, although the models tackle somewhat different tasks, so it is difficult to directly compare. EFBNet would likely perform better on a singular physics engine that would only have variations in a subset of the factors that we randomized. For real world data, there was some promise shown in the ability to predict the direction of the trajectories. Lastly, we developed a robust tool to generate many samples in a simulated environment that provides a lot of information per sample.

References

[1] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016. 2

[2] Peter Battaglia, Tomer Ullman, Joshua Tenenbaum, Adam Sanborn, Kenneth Forbus, Tobias Gerstenberg, and David Lagnado. Computational models of intuitive physics. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, 7 2012. 2

[3] Kiran S. Bhat, Steven M. Seitz, Jovan Popović, and Pradeep K. Khosla. Computing the physical parameters of rigid-body motion from video. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen, editors, *Computer Vision — ECCV 2002*, Lecture Notes in Computer Science, page 551–565. Springer, 2002. 2

[4] João Costeira and Takeo Kanade. A multi-body factorization method for motion analysis. In *Proceedings of IEEE Inter-*

national Conference on Computer Vision, pages 1071–1076, 1995. 2

[5] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021. 2

[6] Kiana Ehsani, Shubham Tulsiani, Saurabh Gupta, Ali Farhadi, and Abhinav Gupta. Use the force, luke! learning to predict physical forces by simulating effects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 2

[7] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards, 2016. 2

[8] Georgia Gkioxari, Ross Girshick, Piotr Dollár, and Kaiming He. Detecting and recognizing human-object interactions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

[9] Saurabh Gupta, Pablo Arbelaez, Ross Girshick, and Jitendra Malik. Aligning 3D models to RGB-D images of cluttered scenes. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 4731–4740. IEEE, 2015. 2

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 4

[11] Mary K. Kaiser, Dennis R. Proffitt, Susan M. Whelan, and Heiko Hecht. Influence of animation on dynamical judgments. *Journal of Experimental Psychology: Human Perception and Performance*, 18(3):669–689, 1992. 1

[12] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014. 2, 3

[13] Jan Koutník, Juergen Schmidhuber, and Faustino Gomez. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 541–548. Association for Computing Machinery, Jul 2014. 2

[14] Nikolaos Kyriazis, Iason Oikonomidis, and Antonis Argyros. Binding computer vision to physics based simulation: The case study of a bouncing ball. In *Proceedings of the British Machine Vision Conference*, pages 43.1–43.11. BMVA Press, 2011. http://dx.doi.org/10.5244/C.25.43. 2

[15] Sascha Lange, Martin Riedmiller, and Arne Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012. 2

[16] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 430–438, New York, New York, USA, 20–22 Jun 2016. PMLR. 2

[17] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 2146–2153. IEEE, 2017. 2

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 3

[19] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 209–217, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 1

[20] Senthil Purushwalkam, Abhinav Gupta, Danny M. Kaufman, and Bryan Russell. Bounce and learn: Modeling scene dynamics with real-world bounces, 2019. 1, 2

[21] Davis Rempe, Srinath Sridhar, He Wang, and Leonidas J. Guibas. Learning generalizable physical dynamics of 3d rigid objects. *CoRR*, abs/1901.00466, 2019. 2

[22] Kevin A. Smith, Kelsey Rebecca Allen, and Joshua B. Tenenbaum. End-to-end differentiable physics for learning and control. *Neural Information Processing Systems (NIPS)*, Dec 2018. Accepted: 2020-08-17T15:06:34Z. 2

[23] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. 3

[24] René Vidal and Shankar Sastry. Vision-based detection of autonomous vehicles for pursuit-evasion games. *IFAC proceedings volumes.*, 15(1):391–396, 2002. 2

[25] Zhihua Wang, Stefano Rosa, Bo Yang, Sen Wang, Niki Trigoni, and Andrew Markham. 3d-physnet: Learning the intuitive physics of non-rigid object deformations. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4958–4964. International Joint Conferences on Artificial Intelligence Organization, 7 2018. 1

[26] Tian Ye, Xiaolong Wang, James Davidson, and Abhinav Gupta. Interpretable intuitive physics model. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 89–105, Cham, 2018. Springer International Publishing. 2