Building Reality Engines

1 Introduction

One of the holy grails in programming is to develop a system that simulates all of reality by representing physical processes across all spatiotemporal scales. If this "reality engine" is accurate enough, it would allow us to develop drug therapies at breakneck speeds, re-create planetary evolution, and dissect the formation and destruction of galaxies. Though this task seems hopeless far off, the last seventy years of computing paint a hopeful picture for the development of such a simulation. In examining this idea, I'd like to provide estimates for the compute required to simulate reality, talk about some problems that are central to computer simulation, and describe recent trends that provide hope for tackling this challenge. In doing so, I'd like to convince you that the seeds for reality simulation are already planted; we are in the nascent stages of re-creating reality in-simulo.

2 Estimating compute for a reality engine

To get a sense of scale, let's try to estimate the compute needed to simulate the entire human body. In describing this system, we are primarily concerned with two questions:

1. At any static moment, how much information is required to encode the state of all particles in the simulation?
2. How much information is required to encode the dynamics function that describes how a state changes over time?

To begin, we know that there are around $7 * 10^{27}$ atoms in the body[1]. If we only encode the 3D position and velocity of an atom in floating point, we would need 24 bytes, meaning that we need $16.8 * 10^{28}$ bytes, or 168,000 yottabytes of data to describe the position and velocity of each atom. For context, it is estimated that all Internet traffic is around 2 zettabytes per year [2], meaning we would need more than 8 million times more data to just describe the static state of the human body.

Next, to describe the dynamics of this system, we would have to iteratively update the unbonded force between each pair of atoms, meaning we would have to perform $168,000^2$ updates. Assuming each update takes on order of 100 flops and we would like to perform an update every femtosecond, we would have to perform $100 * (10^{15}) * 168,000Y^2$ floating point operations to simulate a single second of the human body. This is a hopelessly large number. To make things more realistic, we can assume that each atom only meaningfully interacts with atoms in the millimeter cubed region surrounding it.

In this case, our amount of particles we would have to check against would scale down to $(1.613 * 10^{-8}) * (16.8 * 10^{28}) = 2.71 * 10^{21}$ bytes, meaning we would have to make $(168,000Y) * (.0027Y) = 4.5 * 10^{50}$ checks per update ... which is still an ungodly amount of compute.

These issues with storage and compute only become more egregious as we scale up to trying to simulate something like the entire planet Earth. Because the Earth is around 10^{23} times the volume of an average person, we would have to scale our requirements by that same amount, assuming that we would want to accurately simulate all interactions inside of the planet. Clearly, then, limiting the scope of interactions has some impact on reducing the computational challenge of simulating all pairwise interactions, but it does little to reduce the sheer domain which we have to consider.

3 Core algorithmic problems

Creating a computer system that accurately simulates reality also involves overcoming multiple algorithmic roadblocks: overcoming instability over long timescales, establishing consistency across spatiotemporal scales, and finding efficient approximations for computationally intensive algorithms. To understand how these issues play out at different scales of simulation, I'd like to describe how they've historically played out in three settings: molecular dynamics, weather forecasting, and cosmological simulation.

Presumably, the bedrock of a reality simulator rests at the atomic or subatomic scale, since a large portion of the processes we consider important to human and physical sciences occur at this level. Indeed, many of the greatest advances in biology over the last fifty years have come from simulating molecular processes like ligand binding, protein folding, and drug interactions. This setting also highlights one of the core functionalities that a reality engine would have to possess: the ability to simulate n-body systems with dozens of complex interactions. However, work in molecular dynamics also reveals the limitations of n-body simulation: to accurately model interactions between molecules, computational biologists can't just observe every molecule interacting with one another and develop a simple formula that guides interactions between them. Instead, they develop approximate models called force fields that consist of dozens of experimentally-driven parameters. Historically, the accuracy of these force fields has improved over time [3]. However, these forces are also error-prone and become numerically unstable over long time periods [4].

Furthermore, because these forces have to be calculated over pairwise interactions between all atoms present in a simulation and require a timestep on the femtoscale for accurate simulation [5], even the most successful simulations are limited to the milliseconds range [6].

One alternative is to decrease simulation granularity. In this setting, simulators use aggregate structures to represent groups of atoms, which reduces the number of pairwise interactions that need to be calculated. However, additional work is needed to correctly describe the behavior of each aggregate. And, though various groups have explored methods that are consistent across both the fine and coarse scales [7][8], it is unclear how to best enforce this consistency. This brings up an important decision that designers will have to make when designing reality engines: what is the best simulation granularity and how can approximations be used to ensure spatiotemporal cosistency?

Recently, many have turned to machine-learning methods to solve this approximation problem. In computational biology alone, the number of groups using machine learning to approximate physical processes has exploded over the last ten years [9] and work there has been punctuated by the successes of techniques like AlphaFold, which uses a deep neural network to predict protein structure [10]. However, because processes are strongly coupled across multiple scales, more work is required to determine how to develop consistent methods.

This problem of strong multiscale coupling also shows up in weather modeling. For instance, to perform weather prediction, most simulators have to solve for turbulence in the atmosphere. To make this tractable, they discretize the atmosphere into a grid and perform computational fluid dynamics (CFD) over that medium. However, this ignores the impact of small scale particle movements like eddy formation and modelers have to use sub-grid modeling to correct for these errors [11]. There are two conclusions we can make. On the one hand, this shows that there will always be some level of simulation inaccuracy due to the modeling tradeoffs we have to make in order to make computation tractable. At the same time, work in weather prediction has shown that it is also possible to add error-correction and sub-grid terms to better express the system. However, this comes at the cost of model simplicity and computational efficiency. Naturally, we must ask the same question of a simulation that purports to simulate all of reality: how much error-correction and approximation will be required for a system to run stably?

4 The promise of automated, reductive techniques

Over the course of these last two sections, I've painted a pretty grim picture of the challenges we have to face in creating a reality engine. On the one hand, we have to overcome numerical simulation instabilities, ensure spatiotemporal consistency, and find efficient algorithmic approximations. This leaves an open mystery as to what exact algorithms will be sufficient to simulate real-world interactions. On the other hand, the sheer domain of particles we have to consider severely limits usability.

Faced with these challenges, I believe that the only sensible solution lies in automated, reductive techniques, like those popularized in the machine learning community, that learn both domain representation and dynamics in a more efficient manner. In saying so, I do not imply that the information needed to describe reality can or should be encoded in a series of neural networks. Rather, I believe in some version of the holographic principle, that there exist latent representations that describe reality in a more efficient manner, and that there are automated ways for finding them using different forms of dimensionality reduction, ML models, and deep learning schemes.

To explain why this is important, consider the repercussions of teaching a machine learning model to replace the dynamics function above. If this function learns from real-world data how a population of particles interact, the runtime of this algorithm would be of order N, where N is the number of particles, rather than of order N^2. This would massively decrease the computational requirements for determining each successive dynamics step, but it is an open question whether a "general purpose" dynamics function is both feasible and general enough to replace the kind of manual force-field calculations that are popular today. To make this more concrete, I'd like to close by describing three trends in machine learning that paint a hopeful picture for the future of reality engines.

4.1 Autoencoder-based methods

In both storage and simulation, our biggest problem is dimension: keeping track of yottabytes of information is simply infeasible. One approach is to reduce data dimensionality and instead learn a smaller, latent representation of each simulation state. Indeed, one of the most exciting areas in machine learning is precisely in automated dimensionality reduction; what started with principal component analysis (PCA) and random projection (RP) has morphed into a vast field of diverse methods including autoencoders (AE)[12], variational autoencoders (VAE)[13], generative adversarial networks (GANs)[14], and normalizing flows (NF)[15] that are showing great success in condensing high-dimensionality data like images into latent spaces. Numerous recent works have taken this one step further and attempted to also embed the dynamics function in latent space and have shown early success in model-based reinforcement learning settings [16][17][18]. Although these methods are no where near efficient or mature enough to tackle the simulation size we would require, they point towards a future where such an undertaking is possible.

4.2 Sampling-based learning

In order to learn an accurate simulation, we also have to concern ourselves with the update and evaluation steps: it would be too expensive to evaluate a system on all of its properties at every training step, meaning that we have to sample from the state space instead. Now, sampling-based methods have a rich history and are widely used across all sciences, especially so in ML and robotics[19]. So, in training an accurate simulation, we might imagine that they will continue to play an important role. As one possible application, if a simulation's dynamics are represented as a continuous energy space – where a state's energy dictate how likely the simulation is to "step" into that state –, sampling feasible trajectories of a system through that space will be vital to evaluating the system's performance.

4.3 Graph-based representations

Another important consideration has to do with representation. Namely, what is the best data structure for describing relationships between particle and more granular structures? One interesting approach gaining popular is to represent problems as sparse graphs [20]. In our case, this is a fairly natural representation since we are concerned with learning edge relationships between particles and other structures in a multi-level graph. To this extent, it will be interesting to track the development of graph-based learning methods and see whether they prove to be the most appropriate input representation.

5 Building languages for reality simulation

To close, I would like to speculate on how programmers might interface with reality engines, assuming that the problems above are ameliorated. Once again, we have to reckon with problems of size. For example, if

we were to create a 3D model of the human body, it would take far too long for a single developer to specify all cells for even the simplest organs, let alone to describe the dynamic going-ons within them.

More likely, specifying a simulation would involve little to no manual modeling - having a modeler and animator create a simulation as they might for an animated movie - and instead learn modeling information by ingesting data from real-world observation. For example, to learn a sufficient model of the body, such a simulation might use information from MRI scans, biopsies, and microscopic images. In this sense, simulation construction would involve fewer model-specific parameters and more problem-specific ones. Another approach might be to only define a single simulation once and allow time for interesting structures to develop in it – a la a virtual Big Bang – then search through it to find phenomena worth studying. Though it might prove useful to hand-define some aspects of modeling and dynamics, then, my personal hope lies in automating these processes away such that developers can behave more like scientists than like programmers.

For such a system to be effective across multiple scales, then, it should have a shared notion of modeling, dynamics, and rendering. Is such a proposition possible, given that much of the field today is dominated by domain-specific approaches and workflows? Historically, we can look to the development of industrial simulation software in the 1950s and 1960s, when programmers had a similar goal of developing simulation software that would be usable in a general-purpose fashion. Namely, at the time, researchers at RAND and IBM were concerned with developing a system that allowed for discrete event simulation in industrial and military settings, leading to the development of the General Purpose Simulation Software and later SIMSCRIPT [21]. Though it is difficult to directly relate this goal to the larger one of developing a language that is adequate for describing all physical processes, it does provide some hope that, through careful design and development, it will be possible to develop a language that allows programmers to interface with reality engines.

Though the path to re-creating reality is paved with seemingly insurmountable difficulties, reality engines are not science fiction. I am incredibly excited to see how simulation technologies develop over the course of our lifetimes. I hope that, after reading this, you are too.

References

[1] Anne Marie Helmenstine. How many atoms there are in the human body, 2020.

[2] Cisco. Mobile visual networking index (vni) forecast 2019, 2019.

[3] Kyle A Beauchamp, Yu-Shan Lin, Rhiju Das, and Vijay S Pande. Are protein force fields getting better? a systematic benchmark on 524 diverse nmr measurements. *Journal of chemical theory and computation*, 8(4):1409–1414, 2012.

[4] Ron O Dror, Robert M Dirks, JP Grossman, Huafeng Xu, and David E Shaw. Biomolecular simulation: a computational microscope for molecular biology. *Annual review of biophysics*, 41:429–452, 2012.

[5] Stewart A Adcock and J Andrew McCammon. Molecular dynamics: survey of methods for simulating the activity of proteins. *Chemical reviews*, 106(5):1589–1615, 2006.

[6] David E Shaw, JP Grossman, Joseph A Bank, Brannon Batson, J Adam Butts, Jack C Chao, Martin M Deneroff, Ron O Dror, Amos Even, Christopher H Fenton, et al. Anton 2: raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 41–53. IEEE, 2014.

[7] C Yang, U Tartaglino, and BNJ Persson. A multiscale molecular dynamics approach to contact mechanics. *The European Physical Journal E*, 19(1):47–58, 2006.

[8] Steven O Nielsen, Preston B Moore, and Bernd Ensing. Adaptive multiscale molecular dynamics of macromolecular fluids. *Physical review letters*, 105(23):237802, 2010.

[9] Marinka Zitnik, Francis Nguyen, Bo Wang, Jure Leskovec, Anna Goldenberg, and Michael M Hoffman. Machine learning for integrating data in biology and medicine: Principles, practice, and opportunities. *Information Fusion*, 50:71–91, 2019.

[10] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[11] Frank J Zajaczkowski, Sue Ellen Haupt, and Kerrie J Schmehl. A preliminary study of assimilating numerical weather prediction data into computational fluid dynamics models for wind prediction. *Journal of Wind Engineering and Industrial Aerodynamics*, 99(4):320–329, 2011.

[12] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[14] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Advances in neural information processing systems*, 3(06), 2014.

[15] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[16] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[17] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[18] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: Deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, pages 7444–7453. PMLR, 2019.

[19] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[20] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[21] David Goldsman, Richard Nance, and James Wilson. A brief history of simulation. pages 310 – 313, 01 2010.