Dynamic Frame Generation Using Machine Learning and Scene Data

Mark Wesley Harris
Dr. Sudhanshu Semwal, CS 5790
University of Colorado
at Colorado Springs
May 8, 2019

***Abstract*—This paper describes my preliminary research and implementation of a system for dynamically generating frames or portions of frame of a given animation.**

I. Introduction

Derp intro.

II. Connection to Complex Systems

Derp intro.

A. Global Phenomenon

Derp.

III. Discussion on Research

Derp intro.

A. Research Process

Looked up current progress for prediction during rendering [1] Found paper on next frame prediction given depth data and previous frames. Decided that a CNN LSTM approach would be necessary to generate sufficient results [2] Looked for papers which covered image classification/recognition [3] Found papers on pose estimation [4] Found papers on PixelCNN and PixelCNN++, methods to generate images using CNN [5, 6] Looked for papers on combining multiple aspects of ML algorithms [7] Found Pose Guided image generation systems [8, 9, 10] Found Spatiotemporal Variance-Guidance paper to help with GAN work [11]

B. Frameblocks

One major research subarea is how to generate information from a 3D scene that is valuable to the proposed Machine Learning architecture; I first posed the question of what data should be included. Serializing every attribute of the scene would be excessive, and take much to long to convert and work with. However, using too little data would not work very well either and could produce poor results. I realized that the data for each object must uniquely represent the objects in the scene. Otherwise there would be too much overlapping data, and the algorithm might never learn the differences between the input frames. There are also no limitations nor expectations placed upon the animator; even when comparing scenes which have the same characters, landscapes, and animations the generated data for each object should accurately represent the properties of that object and that object alone.

I found through reading the implementation details of [8] that Ma et. all used a state of the art pose recognition program to generate poses, used as a mask to give extra input to their GAN model. The program assigned 18 (?) key points for each pose, which Ma et. all used as pose data in their program. What was generated was (an image?). These data are not mapped one-to-one with my problem, however; what I need is for the renderer to use unrendered data as well, to solve the problem of reflective, transparent, or other dynamic surfaces. If the camera is facing a mirror, for example, then the entirety of the scene to be rendered is behind the camera. Therefore, unlike with [8] I found that using images as input would not work for this problem, unless the image abstractly captures the scene relative to the camera. Another obvious issue here is that we are trying to speed up rendering as much as possible; any extra rendering could add too much time to the overall algorithm, making it slower than rendering all frames of the animation thereby defeating the purpose of the proposed architecture. The method in which data is stored must be fast, efficient, and encompass only a fraction of the overall runtime of the algorithm.

C. Buffer Frameblocks

Even if we captured all blocks that changed, how many of those changes would be similar? Depending on the frame rate and how many blocks changed, it could mean that most of the same blocks had similar changes which don't need to be

retrained by the program. Given that a scene could have several seconds, if not minutes, of correlated data at a time, it is only natural to crop out important portions of the scene to use for training. Therefore there must also be a dynamic mechanism to select which frames classify as part of the current ?scene? and of those frames which to use as inputs to train on.

There are many different ways to go about this. However we must first define the problem clearly, so the best way can be chosen. I believe the problem in itself does not involve only the color values of each pixel, but rather the relationship between the pixel and its surroundings. Our purpose is to select the best frame blocks out of the set to use in training the later Machine Learning algorithm. The proposed architecture, as discussed before, is meant to take in one frame block and one scene data per iteration and train an algorithm to recognize the causal relationship between the rendered pixels and the 3D scene such that rendered frame blocks can be produced given scene data inputs. The selection of frame blocks is not only dependent upon if the objects in the scene change between frames, but how much they change and in what ways; just because a frame is keyed does not imply there was any major change. As with complex particle systems, simply existing in the scene is all that is needed to create great change in the rendered output. (This is also why it is so pertinent that the later stages of the algorithm take into account both scene data and rendered pixels.) Therefore we cannot rely on scene data in our selection of frame blocks, this selection must come from the frames themselves. This problem is not data-specific, however, as is the case with the later stages. If the pixel data were converted into a format which showed changes and hid similarities, then it could be possible to train a general algorithm which could be queried for any given problem.

It is for this reason I feel a preparation phase is necessary, where each (i)th frame is processed. The first frame $(i = 1)$ is automatically queried, since it provides a baseline for the rest of the frames. Then for each subsequent frame $i = 2...n - 1$ the respective pixel values for the frame and its neighbors are subtracted from each other and these differences are summed together (perhaps the summation could be capped at 255, in order to save memory). This new composite image contains data on what has changed between frames, and it is to be evaluated by the Phase 1 Machine Learning algorithm to determine if the frame has enough changed values. But we should ask ourselves, is a Machine Learning algorithm necessary at this point? Surely we could just use the ratio of dark pixels to light pixels to make this decision. As stated before we are not only concerned with how much the frame changed overall but how it changed and where it changed. There is a problem with using frames that change too much, especially if the frames could be classified as separate scenes; each scene must be processed separately. Thus the decision of whether or not to use a given frame block for the current scene should be a trained decision, where the training data comes from similar analysis of movie frames where the scenes and changing areas are clearly defined. Another point of consideration is using an LSTM model instead of a regular CNN; would previous frames be necessary to determine the amount of change in the current frame?

I have concluded from further study that a Machine Learning algorithm could be used to locate clusters of differences, thereby maximizing the placement of a frame block for a given area inside of a frame. I should look at example/tutorial Machine Learning architectures that provide this kind of clustering ability. Perhaps I should also look into creating multiple different kinds of clusters, and current cluster datasets available online.

D. Training The Algorithm

What of training the algorithm? What kinds of datasets should be used? How many data should be included in each dataset for effective training to occur? Usually, for a Deep Learning project, small images on the order of 10's of thousands are necessary to create a well-trained algorithm. Where might the training data originate from my project, however? The obvious choice is that it must be a subset of rendered frame data. These frames, used in combination with corresponding scene data, would be used as Ground Truth to train a GAN on how to produce images given scene and random inputs. As described in [8], these types of inputs can be used to create low-resolution images, which are then used as the random data for the second GAN which is responsible for creating high-resolution images. For my research there are also CNN LSTM qualities to consider, since all frames for a given ?scene? are related. Multiple frames could be used as inputs, therefore using frames more than once and getting more use out of the training dataset. However, if the training data originates from the current scene, would there be enough frames to justify the use of such a system? If there is only a single frame rendered previously to be used as training data, the answer is simple; render the other frame from scratch! This

is obviously too little data to support wholesome Machine Learning; then, we must ask, where is the cut-off? What variables could this depend on? Size of the scene? Number of keyframes? Number of vertices, edges, or faces? Materials, textures, and attributes? Oh but surely, couldn't we simply say the complexity of the scene as a whole? Therein lies the problem, for we must first classify what complexity means; there must be some ruler by which to measure the complexity of a scene in terms of tangible data for Machine Learning.

So, given there are enough rendered frames respective to the current scene, how might we use them in training? CNN's employ the powerful technique of (as one might guess) convolution. They are able to extract data from the whole of an image without extra identification. Could a similar technique be applied here? Most certainly, given any small portion of an animated scene, is there anyone who could say where it came from on the screen? Without viewing other portions of the scene as well, comparing their composition, and piecing together the rest of the jigsaw puzzle, there is no way to determine from where on the screen a portion originates. This is by nature of the 3D world; no object has simply one view and one view only, the rotation, position, and scale of any object depends on the camera view itself. Therefore, I feel it very possible to split a single frame into several pieces, each with its own ways of seeing the 3D world around them. Especially with extremely high resolutions; a 32x32 pixel block may take up half of one resolutions, while merely a hundredth of another. It is the higher end of this spectrum for which I wish to apply my algorithm, and so I will assume that the blocks of each frame will be a hundredth if not less of each frame. And what of frame rate in producing the quantity of rendered frames? Framerate does indeed play a large role in how many frames are produced, given how many seconds long the clip of an animation is. 60 fps or higher is quite common for high quality gaming applications, however these would not give enough data to support the proposed Machine Learning methodology since future frames and training time is necessary. Real time applications cannot be assisted through this means. Therefore, we must focus on the standard for animation frame rates. As stated in [], studio standards currently average to * fps (assuming 30), which is roughly half that needed in real time rendering.

Given these new attributes of the dataset, how then do the quantity and quality of renders affect training capabilities? Well we must ask several questions. How high of a resolution must the frame be, how many blocks can exist per frame, and how many frames are there per second; only then can we finally stipulate how many seconds is to be expected by this program. Let's assume that 32x32 pixel blocks is sufficient, and that a frame is roughly 1600x1600 pixels (fairly high resolution). This leaves us with 50x50 blocks per frame, or 2500 created blocks total. Similarly, if we assume there are 30 frames per second and half are rendered to be used as part of the training dataset, that leaves us with 37,500 images per second (or 37.5 per millisecond). Now we can finally conjecture as to how many seconds of rendered data would be worth using this approach, and it appears that 1 second might very well be enough! However there is one more factor which we have not yet considered; how much changes within fractions of a second. In order to be considered as part of the same scene, and similarly have enough merit to train over, an object would have to change very minutely each millisecond. There are certain things which behave according to this logic, such as the movement of living things. However, there are also objects such as particle systems which are intended to give the impression of randomness by changing very rapidly. These two types of motion cannot be classified the same way; it is very likely they will need to be dealt with dynamically! For instance, one might imagine the animation of characters talking inside of a cafe on a rainy day. The frame blocks that render the rain through the window are changing very rapidly, while those that contain the characters are most likely changing very slowly. So why not accommodate this phenomenon by quing frame blocks and their scene configuration data based upon the block itself; unchanging blocks need not be queued for further processing, as this could lead to overfitting of the unchanging blocks and underfitting of the dynamic blocks.

E. Training Analysis

However clustering is a problem better suited for data with multiple unique qualities, much like a problem for n-dimensional space. This data does not represent anything more than the amount of difference between two frame's pixels. The argument can be made that there may be waste in the frame blocks; take for example if a frame block edge straddles an area that just barely meets the criteria for processing while neither frame block will be processed. So a set configuration is ineffective for selecting frame blocks, since there may be missing frames in the selection. What about a sliding window? But then by how much should this window slide by? And what

if there is too much overlap between frame blocks selected? Well if there is too much overlap, too many of the same frame will be selected and the algorithm can overtrain. If there is too little coverage, then frames will be missing. The solution? Perhaps a window that slides by half the width of a frame block, and half the height of a frame block. Should there be any extra rules to decide if a block indeed qualifies? For example, if two blocks are chosen because they have the exact same changes shown in both, should one of them be discarded? And should frame blocks be optimized to show changes in the center instead of on one side? Perhaps the frame should be dynamically chosen based upon threshold, shifted by 1 pixel each time until the frame has enough changes shown. A summation of all pixel values of the frame block could be applicable, and once a frame is found the offset could be set to half the frame block width (and height, when next changing rows).

I also feel that the changes should not be a set value either; perhaps it should be a percentage of the total change overall. Take for example a slowly changing gradient from one color to another (changing over multiple frames). We would like to include this, since it is an important lerp to show in the generated frame, however the changes would be continuous over the entire frame. This is a big problem! How do we manage to only capture the frames we need for each configuration? Perhaps previous frame blocks should be captured and hashed, so that similar frame blocks are not captured. To do this I propose we take the first 5 bits of each color value for hashing (so that the last 8 possible values are ignored). Thus there are 15 bits to hash per pixel, and 32 x 32 pixels. I should research further what type of hash could be done. Some data loss is ok, since we want to disregard frame blocks that are too similar. Then maybe values should be super-sampled (averaged?) in smaller blocks to create a smaller 15 bits x 8 x 8 pixel image to compare against. However how much similarity is acceptable? There are probably too many ways to create the generated 8 x 8 pixel image based off of the given frames. Either way, it is clear that some method of hash could be used to avoid training on repeated blocks.

In summary, the total process would be 1) find the frame block (by so far an unknown method), 2) check frame against previously found frames, 3) generate frame block and add to hash list. When the frame is completely processed, clear the list of hashed frame blocks.

IV. Implementation

Derp intro.

A. Frame Processing

The process for generating frameblocks takes in an input of 2 images and outputs all frameblocks which meet our image processing requirements. Given these two image inputs, we must decide what portions of them are different enough to be used as frameblocks.

For this problem, we will need to process every frame with its neighbors. Luckily, the left neighbor will have already been calculated in the previous step. So we only need to compare each frame to its right neighbor for frameblock generation.

Now we must read in the frames and process them. Boundaries are also chosen here, in case the images are different sizes. In theory for this wouldn't occur since the frames of an animation are always the same dimensions, however it doesn't take much time to check. Below the two images are plotted side by side, as a reminder of how different they appear. In theory these images should be two consecutive frames, which usually means they will have very small differences.

B. Shadow Image Creation

Here the pixels of each image are processed. A summation of pixel values is recorded while the loop is running; each pixel adds it's red, green, and blue (RGB) values together, caps the value at 255, and then adds this to the pixel_sum variable.

My goal is to find pixel $P(x, y)$, which represents the difference between the two input pixels at the coordinates (x, y) for each image. I call the image created out of the resulting pixels the "shadow image" for the two inputs. For our purposes, let $p_i(x, y)$ represent the pixel at coordinate (x, y) for image i and let $\oplus$ represent the XOR operation. Thus for each pixel of the input images we calculate the following value:

$$P(x, y) = p_1(x, y) \oplus p_2(x, y)$$

and store this as the value for the output image. Below is the calculated output image. This image graphically shows the differences between frames, where white represents no difference and black represents the maximal difference. We are able to use this later to calculate frameblocks.

C. Frameblock Selection

We select frameblocks using the above calculated image. Frameblocks are selected based on the percentage of black pixels (which in this case represents maximal difference). To calculate the percentage of black pixels, the total sum of pixel values in the shadow image is divided by how many black pixels are possible in the image. Thus the value represents the probability of encountering a changed pixel between the two input images. If the ratio calculated in this way is very low, that means there are very few black pixels so we need to lower our standard for selecting frameblocks. Similarly, if the ratio is close to 1, that means there are many more black pixels and we should raise the standard for selecting frameblocks. This value is directly tied to the input frames.

We need a sliding window to process frameblocks, since we want to skip as many pixels of the same image as possible. The overlap for each frameblock horizontally and vertically is given from the block_offset variable set in the beginning of the program. A value of 1 means no overlap is possible, a value of 2 gives 50% possible overlap, and so on.

Each frameblock selected is saved to an image file. A red rectangle is also overlaid on top of the original calculated image to represent where each frameblock was selected from. Frameblocks which are found to not show enough changes to be included in this iteration are used to find differences spanning multiple frames (described later).

D. Buffer Frameblocks

We stored the frameblocks that failed to pass the test this iteration in two formats: an inverted shadow image file and a frame image file. The shadow image file contains the calculated pixel_sum for each pixel of the ROI of the original shadow image, added together with any previous buffered frameblocks. Thus each frameblock is not only a comparison between the current and next frames, but instead a comparison of all the frames which didn't pass before. It is clear to see from this that even if the change is gradual, it will eventually be represented in the training data. The ROI from the first input image is also saved so that this change can be accurately captured. If the frameblock never changes enough over the entire course of frameblock generation, it is not included in the dataset. An example pair of inverted shadow and frame buffer images is shown below.

E. Data Analysis

So, we were able to generate frameblocks, but how are we to know they are indeed all correct? I propose we compare these outputs to a different scene. As it so happens, the scene processed above is much more complex than we might have realized before. Each sphere is transparent and very reflective, so the changes to one sphere impact changes to all the spheres in the scene. The process used above is repeated below, but for spheres that are completely opaque and diffuse; thus the algorithm is able to filter out more data and choose only frameblocks which change between the two frames.

F. Conclusions

The process shown here is an effective means of choosing frameblocks for training the proposed Machine Learning algorithm. By carefully deciding which portions of the frame to use, we guarantee there will be minimal over-training data and maximum changes shown in the dataset. The next steps of this process will be to develop the Machine Learning algorithm itself, which will take as input the frameblocks of every other frame in an animation.

We have solved the problem of what input would be appropriate to train the proposed Machine Learning algorithm; the dataset generated should contain enough images with enough differences between them. The other input (i.e. the scene data) has not been decided yet. Deciding what aspects of the scene to bundle with each frameblock will require more research and experimentation.

V. Introduction

VI. Ease of Use

A. Maintaining the Integrity of the Specifications

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] Alexander Wilkie, Andrea Weidlich, Marcus Magnor, and Alan Chalmers. 2009. "Predictive Rendering." In ACM SIGGRAPH ASIA 2009 Courses (SIGGRAPH ASIA '09). ACM, New York, NY, USA, Article 12, 428 pages.

[3] R. Mahjourian, M. Wicke and A. Angelova, "Geometry-Based Next Frame Prediction From Monocular Video". 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, 2017, pp. 1700-1707.

[4] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2017. "Show and Tell: Lessons Learned from the 2015 MSCOCO Image Captioning Challenge." IEEE Trans. Pattern Anal. Mach. Intell. 39, 4 (April 2017), 652-663.

[5] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, Dieter Fox. “PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes.” CoRR abs/1711.00199 (2018): n. pag.

[6] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. 2016. “Conditional Image Generation With PixelCNN Decoders.” In Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS’16), Daniel D. Lee, Ulrike von Luxburg, Roman Garnett, Masashi Sugiyama, and Isabelle Guyon (Eds.). Curran Associates Inc., USA, 4797-4805.

[7] Time Salimans, Andrej Karpathy, Xi Chen, Diederik P. Kingma. 2017. “PixelCNN++: Improving The PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications.” ICLR Conference Paper (2017).

[8] Eunvyung Park, Xufeng Han, Tamara L. Berg, Alexander C. Berg. “Combining Multiple Sources of Knowledge In Deep CNN’s For Action Recognition.” 2016 IEEE Winter Conference on Applications of Computer Vision (WACV) (2016): 1-8.

[9] Liquian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, Luc Van Gool. “Pose Guided Person Image Generation.” NIPS (2017).

[10] Bo Zhao, Xiao Wu, Zhi-Qi Cheng, Hao Liu, Zequn Jie, and Jiashi Feng. 2018. “Multi-View Image Generation from a Single-View.” In Proceedings of the 26th ACM international conference on Multimedia (MM ’18). ACM, New York, NY, USA, 383-391.

[11] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, Pieter Abbeel. “InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets.” NIPS (2016).

[12] Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R. Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In Proceedings of High Performance Graphics (HPG ’17). ACM, New York, NY, USA, Article 2, 12 pages.