Variable photorealistic image synthesis for training dataset generation

V.V. Sanzharov[1], V.A. Frolov[2,3], A.G. Voloboy[3]
vs@asugubkin.ru | vladimir.frolov@graphics.cs.msu.ru | voloboy@gin.keldysh.ru
[1]Gubkin Russian State University of Oil and Gas, Moscow, Russia
[2]Moscow State University, Moscow, Russia
[3]Keldysh Institute of Applied Mathematics, Moscow, Russia

Photorealistic rendering systems have recently found new applications in artificial intelligence, specifically in computer vision for the purpose of generation of image and video sequence datasets. The problem associated with this application is producing large number of photorealistic images with high variability of 3d models and their appearance. In this work, we propose an approach based on combining existing procedural texture generation techniques and domain randomization to generate large number of highly variative digital assets during the rendering process. This eliminates the need for a large pre-existing database of digital assets (only a small set of 3d models is required), and generates objects with unique appearance during rendering stage, reducing the needed post-processing of images and storage requirements. Our approach uses procedural texturing and material substitution to rapidly produce large number of variations of digital assets. The proposed solution can be used to produce training datasets for artificial intelligence applications and can be combined with most of state-of-the-art methods of scene generation.

***Keywords**: photorealistic rendering, procedural generation, synthetic datasets, computer vision.*

1. Introduction

There are two main challenges in the training of artificial intelligence models is data quantity and data quality. Data quantity concerns availability of sufficient amounts of training and testing data. Training modern computer vision algorithms requires image datasets of a significant volume - tens and hundreds of thousands of images for training on static images and an order of magnitude more for animation [1, 2] Also, by data quantity we mean how balanced is the data – are all the different classes, which the model must recognize, represented enough. This can be a significant problem because certain classes can be very rare in the data obtained from real world [3]. Data quality can mean many different characteristics, but one that is especially important for images is accurate markup. For example, if model needs to detect certain objects in an image, then in the training data these objects must be accurately annotated. This is usually done manually or semi-automatically with the help of segmentation tools [4, 5]. Annotating a large image dataset manually is extremely expensive, and often manual marking does not have the necessary accuracy (automated markup options also suffer from insufficient accuracy and have disadvantages).

Using synthetic data (in this case, photorealistic images produced by rendering 3d scenes) can easily solve these problems. The solution to data quantity problem can be achieved by using the algorithms for procedurally setting the optical properties of materials and surfaces (displacement maps), this way it's possible to quickly generate almost unlimited number of training examples with any distribution of objects (and therefore classes) present in generated images. In addition, one can create training examples which are scarce or almost non-existent in "real-life" datasets. For example, emergency situations on the road or in production, military operations, objects that only exist as design projects or prototypes. And second, it is possible to produce pixel-perfect image annotation together with rendered image.

But there are also several drawbacks with synthetic data generation. The main problem is producing 3d scene setups suitable for rendering adequate image dataset. The first part of this problem is to generate a 3d scene layout meaningful placement of objects (3d models and light sources) and choosing 3d models to include in the generated scene. And the second part is setting the optical properties of material models for objects in scene, so that they will mimic real-life objects. Similar problem arises in digital content creation in visual effects and video games industries, where several variations of the same digital asset (such as 3d models, material models, texture, etc.) are created by 3d artists using software tools. However, for large datasets generation it's not feasible to manual produce variations of 3d assets.

There is also drawback associated with time and computational resources required to render dataset with the size of thousands of images.

In this work, we propose an approach aimed at alleviating the latter problem by using procedural texturing and material substitution to produce large number of variations from small set of base digital assets.

2. Related Work

The drawback associated with computational resources can be solved by using fast and simple rasterization-based rendering solutions (usually OpenGL-based) [6], possibly in tandem with global illumination approximation such as ambient occlusion [7].

Of course, rendering of thousands of images or sequences requires significant computational resources. But while in simulators developed for training people in many cases a schematic image of a three-dimensional scene is enough for a person (although it is necessary to provide real visualization time), modern AI systems based on deep neural networks are trained according to different principles. It is important for AI systems to accurately model the data on which training is supposed to be carried out (but there is no real-time requirement). For synthetic data to closely approximate real-life

datasets, it should simulate reality (i.e. the image should be photorealistic). Otherwise, there is no guarantee that AI will "work" on similar examples in reality, since it is currently practically impossible to understand the causes of a failure in a multilayer neural network [8-10]. So, photorealistic rendering, which is usually based on path tracing algorithm or its many variants needs to be used. Works [11-13] demonstrate advantages of using photorealistic rendering for synthetic datasets generation. Because the computational cost of physically correct rendering is still quite high and rendering speed and scaling of the training dataset generation system as a whole is important, solutions, relying on photorealistic rendering have disadvantage in this regard. It is, however, alleviated by the recent advent of publicly available hardware accelerated ray-tracing, which can provide significant speedups for photorealistic rendering [14] as well as denoisers [15].

Several approaches exist to automate the process of creating 3d scenes for photorealistic image datasets creation. In [16-19] authors use Augmented Reality (AR) based techniques to insert synthetic objects in photos. This approach requires a way to choose the position of inserted objects – random with some distribution, using existing image annotation or additional reconstruction tools.

In [8] the 3d scene is generated by a set of rules which make use of randomized parameters to select some of 3d models from a database and to procedurally generate others. A similar approach, called Domain Randomization (DR) is used in [20-21]. Domain randomization implies making selection of parameters (aspects of domain) which are randomized for each generated sample. Such parameters may include camera position and field of view, number of objects, number of lights, textures used for objects, etc.

In [22] physical simulation is used to achieve realistic placement of 3d models on a surface.

There are also works that use variety of approaches to scene description and generation, such as domain specific languages [23], scene graphs [24], stochastic grammars [25] for scenes description and generation.

Finally, there are solutions that can generate a whole synthetic dataset similar to specified real-world dataset [26]

These works mainly focus on composing realistic 3d scenes from existing digital assets – 3d models, textures, materials, etc. While in some cases [8] the digital assets itself are randomized, this is done in a very limited manner, -usually only the base material color is changed. And because of it, these approaches require large databases of digital assets to produce images with high variability of objects in them.

One of the methods to further increase variety and realism of synthesized images and to match them more closely to real-life datasets is domain adaptation [27-28]. However, such techniques require additional image processing stage which requires significant time and computational power, especially for images with relatively high resolution.

In this work, we propose an approach based on combining existing procedural texture generation techniques and domain randomization to generate large number of highly variative digital assets during the rendering process. The proposed solution can be combined with most of the reviewed methods of scene generation.

3. Proposed solution

The motivation behind our solution is to produce many variants of the same digital asset (in particular, 3d model with assigned materials and textures) to minimize the amount of manual and expensive work done by 3d artists. To achieve this, we propose the following generation pipeline (fig. 1).

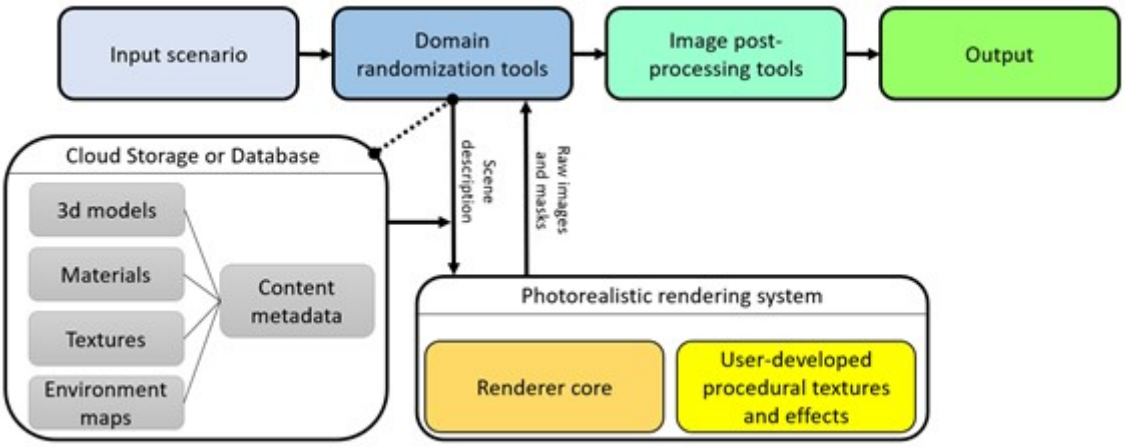

Fig. 1. Architecture of the proposed image generation pipeline

Input scenario specifies settings for the whole pipeline: what kind of scenes are to be generated – classes of objects to be included, lighting type (indoor or outdoor, day or night, etc.), which AOVs (arbitrary output values) should be output by rendering system (instance segmentation masks, binary object masks, normals, depth, etc.), image post-processing (if any) to be done after rendering and randomization domain – which parameters should be randomized and what is randomization distribution - material model parameters, procedural textures and effects, object classes distributions, object placement and so on.

Cloud storage or database contains base digital assets:

1. 3d models with material markup - i.e. what parts of the model have or can have different material types.
2. Materials – base material types, representing common BRDF blends, such as purely diffuse materials (such as brushed wood or rubber), reflective materials (such as polished metals), diffuse + reflective materials (such as plastics or brushed metals), reflective + refractive materials (such as glass), diffuse with two reflective layer (such as car paint with coating) and so on.
3. Textures – collection of image textures and normal maps to be used in materials.
4. Environment maps – HDR spherical panorama images for use for image-based lighting, representing variety of lighting conditions.
5. Content metadata – information that is used by domain randomization tools to select fitting digital assets from the storage according to input scenario. This includes:
   - correspondence of classes to 3d models (for example, which 3d models are models of cars, chairs or humans),
   - correspondence of material classes to materials in the storage (for example, that stained glass and clear non-refractive glass are both of type "glass" and therefore can be assigned to a 3d model part marked as "glass" type),
   - correspondence of textures to material parameters (which textures can be used for which material parameters),
   - information of HDR images (what lighting conditions this particular image has) and so on.

Domain randomization tools produce scene descriptions from input scenario. This stage can query digital assets storage and using content metadata randomly or deterministically (depending on input scenario) select appropriate digital assets and generate requested number of scene descriptions. The generated scene description is intended to be used by rendering system directly.

As **photorealistic rendering system** in our work we used open-source system Hydra Renderer [29] which uses.xml scene description. Scene description also includes what procedural effects should be used and what are their input parameters (if any). Hydra Renderer supports user extensions for procedural textures [30] and the usage of this functionality is one of the key elements of our solution.

There are several properties of procedural textures that make them a vital element in our generation pipeline:

1. Procedural textures can be parametrized with arbitrary values and therefore it is possible to generate a large number of variations of the same texture.
2. It's possible to apply texture to geometry without uv-unwrapping if the texture is parametrized by, for example, world space or object space position. This allows to relax requirements for 3d models and eliminate predominantly manual work of doing uv-mapping for them.
3. Finally, procedural textures don't have fixed resolution (resulting texture is infinite and has no seams) and because of this it is possible to produce detailed high-quality materials suitable for application to variety of 3d models of different scale.

As a part of this work we developed several procedural textures which allowed us to greatly increase variation of 3d objects and also increase realism of their appearance.

Image post-processing tools goal is to adjust images, output by the rendering system or produce additional data about these images. The tasks performed by this stage can involve:

1) measuring 2d bounding boxes for objects/instances;
2) applying variety of image-space effects to further increase variety of output images or better match them to real-life datasets, for example:
   - chromatic aberrations,
   - barrel simulation,
   - blur,
   - transformations and warping the image, including resampling for the purpose of anti-aliasing,
   - noise
   - and others.
3) cutting objects out of rendered image,
4) composing rendered objects with other images (as Augmented Reality based solutions mentioned earlier do)
5) format conversions,
6) and others.

It is worth noting that all listed tasks can be performed using simple python scripts or open-source compositing software like Natron [31,32] and don't need complex and computation-intensive processing with neural networks.

In the described image generation pipeline architecture, the domain randomization tools stage can be replaced by any other of the reviewed approaches to scene generation – scene graph produced by neural network processing of existing datasets, stochastic grammars, markup data from existing dataset for employing augmentation reality techniques. Or any custom scene generation solution, for example placing objects inside existing scene with respect to its depth buffer.

4. Object appearance variation techniques

Procedural textures

As we mentioned before, one of the key parts of our work is the use of procedural textures. In this section we describe procedural textures developed for use in our generation pipeline.

The first problem we were trying to solve with procedural textures is to provide additional details to rendered 3d models to produce more realistic images in contrast to crisp and clear look of rendered objects. For this purpose we implemented several procedural textures, simulating effects such as dirt, rust and scratches on materials textures (fig. 2-5).

Fig. 2. Rust procedural texture variations on different models

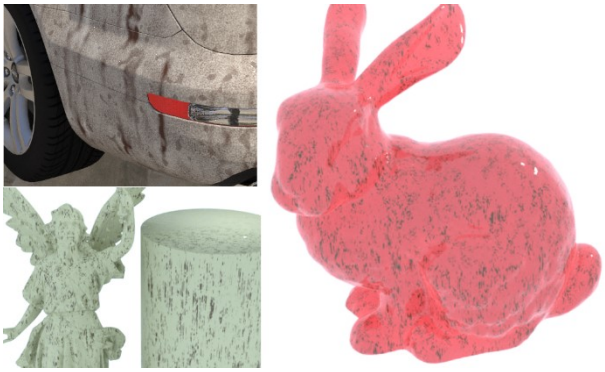

Fig. 3. Dirt procedural texture variation on different models

Fig. 4. Scratches procedural texture variations. Also affects normal map

Fig. 5. Rust and dirt procedural textures applied to road signs models normal map

All these textures were parametrized in a way, that allowed domain randomization tools significantly vary the appearance of the texture by passing different (and possibly random) values as these parameters. Since the implementation of these procedural effects is predominantly based on noise functions most of these parameters correspond to noise parameters such as amplitude, frequency and persistence. Among other common parameters used is the relative height (or other dimension) of an object, the effect reaches. This allows as to dynamically control how far effect spreads on 3d model, which is impossible with ordinary textures.

Developed procedural textures can affect not only colors or blending masks between different materials, but also normal maps (fig. 6) or can be used for displacement maps to slightly deform the object (fig. 7). Changing geometry in this way also produces changes in its silhouette and therefore in segmentation masks.

Fig. 6. Procedural displacement. Top - without displacement, bottom - with mild displacement, warped regions are highlighted

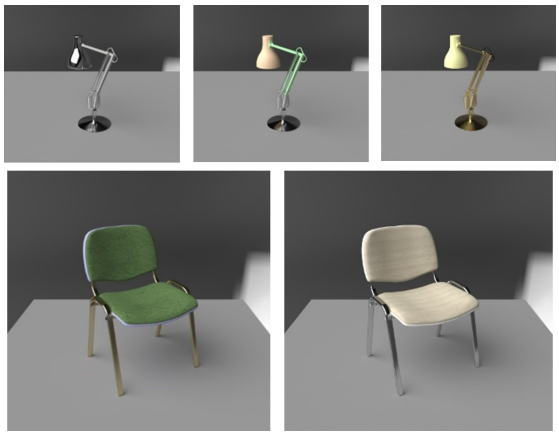

Fig. 7. Material substitution example

Material substitution

In addition to procedural textures, another technique to increase variability of 3d models implemented in proposed solution is material substitution. In the proposed data generation pipeline, digital content storage contains materials, while 3d models are marked up with material types. This allows to specify a collection of materials – manually created, pre-generated or imported from one of the existing open libraries. These materials can then be classified into

several categories such as “wood”, “metal”, “car paint”, “plastic” and so on. And during scene generation phase, domain randomization tools can allow 3d models to use random materials within classes, specified as possible for this model. For example, chairs can have materials from “wood” or “metal” classes, while “glass” type materials are unlikely to be assigned to chair model.

In the reviewed existing solutions this technique is mostly used in a very rudimentary form – only color is changed, not the material (i.e. BRDF or BRDF blend) itself.

5. Results and conclusion

Proposed image datasets generation pipeline architecture can create many variations of the same 3d model using procedural textures and material substitution techniques. Other 3d scene parameters, which are commonly varied in existing solutions, such as lighting (HDR spherical maps for image-based lighting point and area lights) can also be utilized in the proposed pipeline. This allows to use much smaller pre-existing digital assets collection while producing output images of high variability.

Let’s consider the case of a simple scene with only one 3d model in it. In existing solutions appearance of a single 3d model is most commonly varied by a single parameter – base color. Proposed solution allows also to specify several procedural textures, each with at least 3 parameters (related to noise functions), which significantly alter the appearance of objects (fig. 2-5). So, for single 3d model with single material each procedural texture introduces another 3 dimensions of appearance variability compared to only single “color” dimension in existing solutions. This allows our solution to produce exponentially more variations of a single 3d model with single material. And with material substitution we can also alter the material types suitable for this particular 3d model.

Existing works on synthetic image datasets generation are mostly concerned with scene generation and rely on having large collections of digital assets to construct these scenes from. While there exist large 3d models collections such as ShapeNet [33], they usually have content of poor quality compared to assets, created by experienced 3d artists. And with proposed approach of using procedural textures and material substitution, it is possible to produce many variations out of small set of high-quality models that portray real-life objects more accurate.

However, proposed solution doesn’t exclude lower quality models and is able to deal with 3d models without texture coordinates because of used procedural texturing techniques.

Finally, proposed image generation pipeline can be integrated with any of the reviewed solutions for scene generation.

6. References:

[1] Karpathy, Andrej, et al. Large-scale video classification with convolutional neural networks. // Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2014

[2] Wu, Zuxuan, et al. Deep learning for video classification and captioning // Frontiers of multimedia research. 2017. 3-29.

[3] Фаизов Б.В., Шахуро В.И., Санжаров В.В., Конушин А.С. Классификация редких дорожных знаков // Компьютерная Оптика, Т. 44, №2, 2020

[4] Moehrmann, Julia, and Gunther Heidemann. Efficient annotation of image data sets for computer vision applications. // Proceedings of the 1st International Workshop on Visual Interfaces for Ground Truth Collection in Computer Vision Applications. 2012.

[5] Gao, Chao, Dongguo Zhou, and Yongcai Guo. Automatic iterative algorithm for image segmentation using a modified pulse-coupled neural network. // Neurocomputing 119 (2013): 332-338.

[6] Su, Hao, et al. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. // Proceedings of the IEEE International Conference on Computer Vision. 2015.

[7] Kirsanov, Pavel, et al. DISCOMAN: Dataset of Indoor Scenes for Odometry, Mapping And Navigation. // arXiv preprint arXiv:1909.12146 (2019).

[8] Nguyen, Anh, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. // Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[9] Zhang, Chiyuan, et al. Understanding deep learning requires rethinking generalization. // arXiv preprint arXiv:1611.03530 (2016).

[10] Montavon, Grégoire, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks // Digital Signal Processing 73 (2018): 1-15.

[11] Movshovitz-Attias, Yair, Takeo Kanade, and Yaser Sheikh. How useful is photo-realistic rendering for visual learning?. // European Conference on Computer Vision. Springer, Cham, 2016.

[12] Tsirikoglou, Apostolia, et al. Procedural modeling and physically based rendering for synthetic data generation in automotive applications. // arXiv preprint arXiv:1710.06270 (2017).

[13] Zhang, Yinda, et al. Physically-based rendering for indoor scene understanding using convolutional neural networks. // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017

[14] Sanzharov V., Gorbonosov A., Frolov V., Voloboy A. Examination of the Nvidia RTX // CEUR Workshop Proceedings, vol. 2485 (2019), p. 7-12

[15] S.V.Ershov, D.D.Zhdanov, A.G.Voloboy, V.A.Galaktionov. Two denoising algorithms for bi-directional Monte Carlo ray tracing // Mathematica Montisnigri, Vol. XLIII, 2018, p. 78-100.

https://lppm3.ru/files/journal/XLIII/MathMontXLIII-Ershov.pdf
[16] Alhaija, Hassan Abu, et al. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. // International Journal of Computer Vision 126.9 (2018): 961-972.
[17] Dosovitskiy, Alexey, et al. Flownet: Learning optical flow with convolutional networks. // Proceedings of the IEEE international conference on computer vision. 2015.
[18] Varol, Gul, et al. Learning from synthetic humans. // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.
[19] Chen, Wenzheng, et al. Synthesizing training images for boosting human 3d pose estimation. // 2016 Fourth International Conference on 3D Vision (3DV). IEEE, 2016.
[20] Tobin, Josh, et al. Domain randomization for transferring deep neural networks from simulation to the real world. // 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2017
[21] Prakash, Aayush, et al. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. // 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019
[22] Mitash, Chaitanya, Kostas E. Bekris, and Abdeslam Boularias. A self-supervised learning system for object detection using physics simulation and multi-view pose estimation. // 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017.
[23] Fremont, Daniel J., et al. Scenic: a language for scenario specification and scene generation. // Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2019
[24] Armeni, Iro, et al. 3D Scene Graph: A Structure for Unified Semantics, 3D Space, and Camera. // Proceedings of the IEEE International Conference on Computer Vision. 2019
[25] Jiang, Chenfanfu, et al. "Configurable 3d scene synthesis and 2d image rendering with per-pixel ground truth using stochastic grammars." International Journal of Computer Vision 126.9 (2018): 920-941.
[26] Kar, Amlan, et al. Meta-sim: Learning to generate synthetic datasets. // Proceedings of the IEEE International Conference on Computer Vision. 2019.
[27] Hoffman, Judy, et al. Cycada: Cycle-consistent adversarial domain adaptation. // arXiv preprint arXiv:1711.03213 (2017).
[28] French, Geoffrey, Michal Mackiewicz, and Mark Fisher. Self-ensembling for visual domain adaptation. // arXiv preprint arXiv:1706.05208 (2017)
[29] Ray Tracing Systems, Keldysh Institute of Applied Mathematics, Moscow State Uiversity. Hydra Renderer. Open source rendering system, 2019, https://github.com/Ray-Tracing-Systems/HydraAPI
[30] V.V. Sanzharov, V.F. Frolov. Level of Detail for Precomputed Procedural Textures // Programming and Computer Software, 2019, V. 45, Issue 4, pp. 187-195 DOI:10.1134/S0361768819040078
[31] Natron, Open Source Compositing Software For VFX and Motion Graphics https://natrongithub.github.io/
[32] A.E. Bondarev. On visualization problems in a generalized computational experiment (2019). Scientific Visualization 11.2: 156 - 162, DOI: 10.26583/sv.11.2.12 (Scopus) http://www.sv-journal.org/2019-2/12/
[33] Chang, Angel X., et al. "Shapenet: An information-rich 3d model repository." arXiv preprint arXiv:1512.03012 (2015).

About the authors

Vadim Sanzharov, senior lecturer, Gubkin Russian State University of Oil and Gas. E-mail: vs@asugubkin.ru.

Vladimir Frolov, PhD, senior researcher at Keldysh Institute of Applied mathematics RAS, researcher at Moscow State University.

Alexey Voloboy, D.Sc., PhD, leading researcher at Keldysh Institute of Applied mathematics RAS.