



## A Simulation System for Scene Synthesis in Virtual Reality

Liu, Jingyu; Mantel, Claire; Schweiger, Florian; Forchhammer, Søren

*Published in:*  
Proceedings of EuroXR 2021

*Publication date:*  
2021

*Document Version*  
Peer reviewed version

[Link back to DTU Orbit](#)

*Citation (APA):*  
Liu, J., Mantel, C., Schweiger, F., & Forchhammer, S. (Accepted/In press). A Simulation System for Scene Synthesis in Virtual Reality. In *Proceedings of EuroXR 2021* IEEE.

---

### General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

# A Simulation System for Scene Synthesis in Virtual Reality <sup>\*</sup>

Jingyu Liu<sup>1</sup>[0000-0002-1663-4222], Claire Mantel<sup>1</sup>[0000-0002-3081-7922], Florian Schweiger<sup>2</sup>[0000-0003-2026-7688], and Søren Forchhammer<sup>1</sup>[0000-0002-6698-8870]

<sup>1</sup> Technical University of Denmark, Kgs. Lyngby 2800, Denmark  
{jing,c1ma,sofo}@fotonik.dtu.dk

<sup>2</sup> BBC Research & Development, London W12 7TQ, United Kingdom  
florian.schweiger@bbc.co.uk

**Abstract.** Real-world scene synthesis can be realized through view synthesis or 3D reconstruction methods. While industrial and commercial demands emerge for real-world scene synthesis in virtual reality (VR) using head-mounted displays (HMDs), the methods in the literature generally do not target specific display devices. To meet the rising demands, we propose a simulation system to evaluate scene synthesis methods in VR. Our system aims at providing the full pipeline of scene capturing, processing, rendering, and evaluation. The capturing module provides various input dataset formulations. The processing and rendering module integrates three representative scene synthesis methods with a voluntary performance-aid option. Finally, the evaluation module supports traditional metrics as well as perception-based metrics. An experiment demonstrates the use of our system for identifying the best capturing strategy among the three degrees of foveation tested. As can be expected, the FovVideoVDP metric (based on a model of the human visual system) finds the highest degree of foveation giving best results. The three other quality metrics from the evaluation module (which use features to measure similarity) confirm that result. The synthetic scenes in the experiment can run in VR with an average latency of 5.9 ms for the two selected scenarios across the tested methods on Nvidia GTX 2080 ti.

**Keywords:** View synthesis · 3D reconstruction · Virtual reality

## 1 Introduction

In recent years, virtual reality (VR) as a novel display and interactive platform has made many experimental manifestations possible in the field of media. Synthesis of real-world scenes in VR with high-fidelity, for example, is an active research topic. A complete scene synthesis pipeline for VR includes steps of data

---

<sup>\*</sup> This project has received funding from the European Union’s Horizon 2020 research and innovation programme under Marie Skłodowska-Curie Grant Agreement No. 765911 (RealVision). Special thanks to BBC Research & Development for providing the dataset and the consultancy of scene synthetic methods.

capturing of the real-world scene, data processing, and synthetic scene rendering in VR. The literature on 3D reconstruction or view synthesis usually covers only a part of the pipeline, furthermore the methods are relatively general-purpose and lack adaptation and/or optimization for VR. In addition, it is complex to a priori identify the best scene synthesis method for a specific application scenario, due to the lack of VR-tailored evaluation mechanisms. Thus in practice, to have a VR-tailored integrated solution for scene synthesis is an open question.

To access this question, in this paper, we present a scene synthesis system for VR through a simulation test-bed built upon a game engine (Unreal). The system aims to provide a solution for users to screen various scene synthesis methods on the dataset to be tested. A virtual capturer in our system can help the users with dataset simulation (which will generate computer graphics images), users can also capture dataset as real-world photographs using cameras and input into our system. A VR-tailored evaluation module in our system can score the methods to provide users with perception-based objective assessments.

The structure of this paper is as follows: Section 2 summarizes scene synthesis methods in the literature. Section 3 designs the system with the following four aspects in mind: 1. Input dataset with different features, 2. Geometric proxies and their pros and cons, 3. Texturing and rendering methods with an adaptation and/or optimization for VR, and 4. Objective metrics and their feasibility across various geometric proxies. Section 4 details the implementation of the system for the four major modules: capturing, processing, rendering, and evaluation. Section 5 describes an experiment that demonstrates an application of our system: evaluating the quality for different scene synthesis methods in regard to the capturing strategy for the input dataset. (The simulated input datasets for the experiment are computer graphic images). Finally, Section 6 discusses the system with a future work. An illustration of the pipeline is depicted in Figure 1.

The main contributions of the paper are:

- A VR-tailored simulation system integrates a whole pipeline of scene synthesis of capturing, processing, rendering and evaluation, with adaptations and/or optimizations to the selected representative scene synthesis methods.
- An experiment that gives objective scores to compare different capturing strategies for the input dataset for the scene synthesis methods with four quality evaluation metrics, including a HVM-based metric FovVideoVDP.

## 2 Related Work

### 2.1 Scene Synthesis Methods

In this paper we choose to categorize the scene synthesis methods according to the way of processing the geometry of the scene. Since the subsequent rendering method for the view synthesis will be affected by the geometric proxy. Thus the methods are subdivided into three categories: without, explicit and implicit geometric proxy restoration. Richardt et al. [22] updated a similar categorization in the form of continuum which was proposed by [25]. The synthesis approach

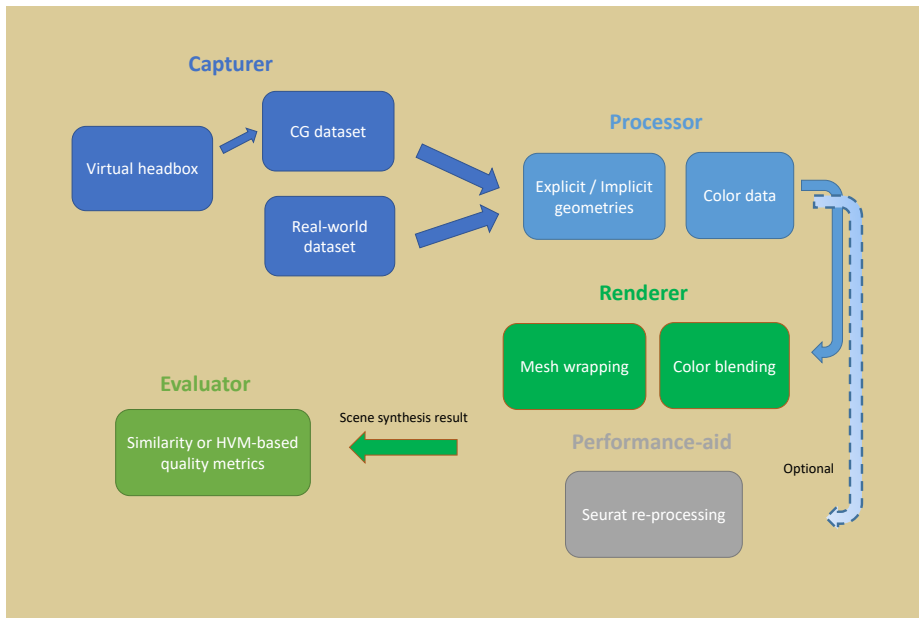


Fig. 1: Pipeline illustration of our simulation system.

without geometric proxy represents a collection of pure image-based rendering (IBR) methods. Explicit geometric proxy reconstruction means that the geometric information is presented in the form of a 3D mesh, i.e. a collection of point cloud or triangles, while implicit geometric proxy restoration means that the geometric information is stored in an intermediate data structure. While different geometric proxies can sometimes be convertible, the purpose of this categorization is mainly to guide the design and optimization of different processing and rendering pipelines in Section 3.

A typical scene synthesis method without geometric proxy is through *light field* [11] or *luminograph* [7] rendering. This IBR method realizes purely through view synthesis. Another improved version of light field rendering is proposed in [2] as unstructured light field rendering (ULR). In general IBR-based methods suffer from artifacts that they can not align objects with different depths. Panorama or omni-directional stereo (ODS) is another solution without geometric proxy widely applied in VR scene synthesis. However, it only has three degree of freedom (DoF) without parallax. Luo et al. [14] and Bertel et al. [1] brought parallax into ODS, however the enhanced motion parallax is limited.

*Photogrammetry* [5] refers to the solutions with explicit geometric proxy. The advantage of the photogrammetry is that once the geometry of the scene is reconstructed, users truly have 6 DoF to observe novel views in the reconstructed synthetic scene. However, to output a high-definition synthetic scene, photogrammetry needs to create models with dense meshes, which is challenging

in practice. For instance, Quixel [9] builds a single asset with around 1 million polygons. It means that the polygon budget of a reconstructed scene will be of the order of billions, i.e. far more than the overhead of VR rendering. *COLMAP* [24][23] is another photogrammetry method that draws upon structure-from-motion (SfM) and multi-view stereo (MVS). Compared to Quixel, COLMAP sacrifices the accuracy of the geometric reconstruction in exchange for time efficiency. Artifacts can be seen as holes and seams in the reconstructed 3D meshes. Another representative method to reconstruct explicit geometry is unstructured point clouds. This data structure is an intermediate state of COLMAP pipeline. However, in other implementations, together with color information, point clouds can also be the final explicit geometry output of 3D reconstructions [27].

Implicit geometric proxy restoration methods, such as per-view geometry [8][19], multi-plane images (MPI) [31][4][17], voxels [13] etc, is worth an extended discussion. The reason is that it is more consistent with the goal we want to achieve: scene synthesis in VR with high-fidelity novel views, but not accurate geometries. Thus the geometric proxy should be treated as a medium, not a target. Overbeck et al. [19] proposed a method that is light-field-based with an implicit local geometric proxy generated and stored. Note that as an inheritor of the light field, [19] made a special-designed camera rig as the capturer to retrieve a dense dataset with thousands of images. Mildenhall et al. [4] alleviates the data density by utilizing machine learning: with a learned gradient descent, the implicit geometric proxy is represented as MPI. The method even supports photo-realistic reconstruction for volumetric effects like fire with only a sparse light field dataset. In addition to still scenes, dynamic scenes reconstruction is also supported with an upgraded multi-sphere images (MSI) geometric proxy. However, novel views can only be synthesized over a limited baseline, i.e. an inside-out experience will be hard to achieve. Mildenhall et al. [18] utilizes volume as an alternative to voxel as its geometric proxy. It defines 5D neural radiance fields (NeRF) (3D for location and 2D for direction), which can handle complex materials, detailed geometries, and multiple depths. Wizarawongsa et al. [29] (NeX) supports view-dependent effects that are derived by reflectance coefficients adapted in its model, while the method can realize view synthesis in real-time. Note that while neural network-based geometric proxies can provide better synthesis quality, they are not necessarily the best choice for VR. Since the information of the trained geometry is stored in a checkpoint structure. On the one hand, reading the information from the checkpoint and then generating a synthetic view may not be in a real-time manner (such as NeRF). On the other hand, even if the method can generate synthetic view in real-time from the original network (such as NeX). There is no reliable solution to directly process the result to VR displays.

## 2.2 System Integration and Methods Evaluation

In addition to the discussion of scene reconstruction methods, Dupont et al. [3] provides a toolkit from the perspective of system design, which is a good

reference for our work. However, the design of the evaluation module of the system is relatively simple and can be further improved.

Pixel-based image quality metrics such as PSNR is still applied in the latest scene synthesis methods such as [29]. However, for scene synthesis applications, perceptual image similarity metrics are required - as suggested by [12] - which analyze more the high-level similarity (or structural similarity) between images, rather than pixel-to-pixel differences.

In addition, there is no preference for specific display devices when comes to evaluation of the scene synthesis methods in general. However, whether a direct application of traditional metrics can really reflect the perceptual quality for scene synthesis in VR is worth questioning. VR devices are very different from desktop displays regarding to parameters such as lens distortion, resolution, field of view (FoV), distance from the screen, and shielding of ambient light sources. Especially, due to the large FoV, the visual acuity feature in the human visual model (HVM) is worthy of attention in perceptual quality. As it is focused on visualization on HMDs through both display modeling and inclusion of differentiated center/periphery aspects in the HVM, the recent FovvideoVDP metric from [15] is very relevant.

### 3 Design

To design a good simulation system, various factors should be considered: 1. What scene capturing patterns should be included? 2. How different geometry proxies can adapt to the platform? 3. What rendering methods should be implemented? 4. What metrics are geometric proxy independent and can be VR-tailored? etc. In this section, the questions above will be addressed.

#### 3.1 Data Capturing

Realizing a virtual capturer is the main focus for our capturing module design, which simulates a real-world camera set up. Considering the real-life application scenarios, limited data types can be obtained from commercial cameras, of which a vast majority produce RGB images. Additionally, as a test-bed, we can also generate a ground truth depth map for users' reference and calibration.

When designing the set up of the virtual capturer, on the one hand, it should meet the needs of an immersive VR experience (i.e. an inside-out experience for the major cases). On the other hand, users should be able to toggle the configuration of the capturer (i.e. camera parameters, capturing density, capturing pattern, etc). In our design, we use a *datapoint* to represent a piece of capturing, and we call the region in a scene that datapoints distribute and cover as *headbox*. To support various scene synthesis methods, a datapoint can be either a cube-map or an image, with or with out a corresponding depth map. The configuration of the virtual capturer will reflect the complexity of the setup in real-life. For example, a uniformly distributed sampling implies high calibration requirements while a random distributed sampling represents a more casual setup. Besides,

to simulate the photograph as much as possible, we need to make full use of a high-definition rendering pipeline in the synthetic test-bed. Features such as ray-traced ambient occlusion, reflection, and refraction should be enabled for capturing to restore geometry, lighting, and material information of the scene.

### 3.2 Geometric Proxies

Different geometric proxies have each their advantages and disadvantages, and we must especially consider their applicability in VR. While an explicit geometric proxy stored in a data type such as .obj is natively supported by most game engines, usually implicit geometric proxies need to be further converted into application-friendly data structures to utilize game engines' VR-ready rendering pipeline, and corresponding rendering strategies are required.

We first consider improving a photogrammetry method and adapted it to our system. COLMAP is a good baseline that has been widely applied due to its open-source and continuous maintenance. Note that we only use partial of the functions, which are the camera poses restoration from SfM and depth estimation and mesh reconstruction from MVS. The explicit global 3D mesh and implicit per-view meshes can be branches from the utilization of the COLMAP modules (i.e. the explicit global 3D mesh is reconstructed from MVS, with the fusion to generate point cloud that eventually turns to a 3D mesh from triangulation; The per-view meshes are reconstructed from the per-view depth maps from local depth estimation with triangulation). Note that we only use COLMAP for the reconstruction of geometric proxies, for texturing we will bring up an optimized solution (in Section 3.3). In addition, we can make a calibration based on the features of our simulation system (we have the ground truth of camera poses and scene depth) on top of COLMAP's solution.

A state-of-the-art neural-network-based view synthesis method that supports view-dependent effect pretty well is NeX[29], which we choose to implement in our design. However, as mentioned the neural network model cannot be directly streamed into VR. Therefore, in order to represent the implicit geometric proxy that is trained from neural network and stored in the form of checkpoints, we need to do some conversion. The detailed implementation is introduced in Section 4.3.

### 3.3 Rendering Methods

First we can discuss the texturing method for explicit 3D meshes. A direct texture mapping applied on top of a global mesh can be unsatisfactory since view-dependent effects such as reflections are not supported because they will be averaged out in the textures, in other words, all pixels in the textures are default to be diffuse. Here we consider using a classic multi-view-based blending method ULR to restore the view-dependent effects. However, a straightforward ULR method can be performance intensive for VR as [3] suggests, thus an optimization should be applied.

Disk-based texturing (DBT), originally proposed by [19], is the texturing method we choose for implicit pre-view proxies. As the textures are sampled to the viewing direction, DBT also supports view-dependent reflection and highlight. An extra benefit is that DBT was brought up for VR experience in its original purpose. The disk-shaped texturing window can reduce the per-view texture sampling overhead, which is performance friendly.

In NeX, the MPI geometry and the reflectance coefficients it adopts bring in unique advantages in reconstructing transparent surfaces with refraction compared with the methods discussed above. To render the mesh converted from the checkpoints, we need to figure out the color of vertices in the runtime. The detailed implementation is introduced in Section 4.4.

### 3.4 Evaluation

Identifying the relevant objective quality metrics is the focus while designing the evaluation module. The following requirements are considered: 1. Appropriate metrics should be independent from the geometric proxy. 2. Ideally, quality metrics should be VR-tailored to reflect the real experience.

The first requirement has already screened out many scene synthesis evaluation metrics since they focus on evaluating the reconstructed explicit geometries proxies itself. Metrics applied for images and rephotography are found to be better fits. Besides, for VR-tailored evaluation, perceptual image similarity metrics are required, as suggested by [12]. We found that FovVideoVDP [15] is a good fit since it is not only perception-based, but also take features in HVM into consideration. If the conditions are relaxed and not to consider HVM features, MS-SSIM [28] and LPIPS [30] can also be taken into our model. All these metrics were designed to evaluate more the high-level similarity between images, rather than pixel-to-pixel differences such as PSNR, and therefore they are more consistent with our system. Metrics such as 1-NCC discussed in [26] is a good supplement. All those metrics require the registration of the synthetic views with a reference. With the camera poses restored by SfM, the selected ground truth input can be compared with the synthetic view setting to the corresponding camera pose in the evaluation.

## 4 Implementation

In this section, we explain the details of implementing our design into a simulation system. First, we introduce the platform for our implementation. Then we break down the system into four modules: 1. A capturing module that generates a user-defined dataset from synthetic scenes. 2. A processing module that inputs the user-defined data and outputs user-selected geometric proxies. 3. A rendering module that inputs geometric proxies from the processor and outputs synthetic views in VR with the corresponding rendering methods. 4. An evaluation module that takes a random or user-selected reference view, and outputs the score of the corresponding synthetic view. The overview of the pipeline is illustrated in Figure 1.



#### 4.1 Platform introduction

The system has been implemented on top of the Unreal engine. Our implementation utilizes the engine’s libraries for shaders. The reason why we chose Unreal is because it has good built-in support for VR, furthermore, game engines support various 3D model types, which endorse our implementations for geometric proxies. Note that choosing to implement a system on game engines means we rely on the platform’s resource management, which may not always provide the best performance.

#### 4.2 Capturer

Our virtual capturer realized with blueprints in Unreal. We utilized the built-in command-line "High Resolution Screenshot". Note that in our implementation, we have enabled the ray-tracing-related functions in the engine to get the photo-realistic capturing results. We also accessed to depth buffer for ground truth depth map acquisition.

According to the parameters input by the users, the virtual capturer generates a datapoint distribution. Corresponding datapoints will be dumped into the hard disk. The specific parameters that can be set by users are shown in Table 1. The options define the configuration of virtual capturer from different dimensions. Most options function as the name suggests. For datapoints density  $\rho$ , we define a term *cone density*, which means the average angle difference ( $v$ ) between the adjacent two datapoints in either  $\theta$  or  $\phi$  axis in the polar coordinate. The smaller the  $v$  is, the denser the datapoints is. For *pattern*, uniform distribution for a sphere follows a cosine-based hemisphere sampling [20] which applies an inversion method that generates uniformly sampled random values  $\xi_1$  and  $\xi_2$  that fall in the range of the FoV trigonometric (if FoV is set, otherwise the value ranges  $[0, 1)$ ). Then apply the inverse cumulative distribution functions (CDF) to obtain  $\theta$  and  $\phi$ . Finally convert from polar space to Cartesian space. The pattern for the uniform cuboid sampling follows a similar logic, but instead of using inverse CDFs to get  $\theta$  and  $\phi$ ,  $u$  and  $v$  values are obtained, and there is no extra need for coordinate transformation. If the pattern is set to "foveated", we implement a method that introduced in [16] which is kernel based foveation: The random values  $\xi_1$  and  $\xi_2$  stand for the samplings for the two axis in polar space. From  $\xi_1$  and  $\xi_2$  we have Equation 1.

$$\begin{aligned}
 y &= \exp\left(\frac{\log(\sqrt{2})\xi_1^\omega}{2}\right) \cdot \cos(\pi\xi_2) \\
 z &= \exp\left(\frac{\log(\sqrt{2})\xi_1^\omega}{2}\right) \cdot \sin(\pi\xi_2) \\
 x &= \sqrt{1 - (y^2 + z^2)}
 \end{aligned}
 \tag{1}$$

Where  $\omega$  is the kernel parameter indicate how the datapoints spread across correlated to foveation.  $xyz$  are the normalized datapoint location taking  $(0, 0)$

as the origin. When sampling, skip the  $yz$  pairs fall out of the unit circle when it is "sphere" type.

Note the datapoints distribute on the surface of the headbox. For sphere, the datapoints' direction is the outward normal of the spherical surface; For cuboid, the datapoints' direction is the outward normal of the faces.

In principle, by using this capturing module, users can generate datasets as they were camera-captured photographs.

Table 1: Parameters for the main modules

| (a) Capturing module |                             | (b) Processing & rendering module |                                     |
|----------------------|-----------------------------|-----------------------------------|-------------------------------------|
| Parameters           | Description                 | Parameters                        | Description                         |
| $shape$              | cuboid / sphere             | $cali_c$                          | calibrate SfM camera poses          |
| $type$               | panorama/ forward-facing    | $cali_d$                          | calibrate MVS depth map             |
| $depth$              | capture depth               | $th_c$                            | NeX cube marching threshold         |
| $pattern$            | uniform / foveated          | <hr/>                             |                                     |
| $res$                | datapoint resolution        | Parameters                        | Description                         |
| $o$                  | headbox center              | $n$                               | # of textures / meshes for blending |
| $\rho$               | datapoints density by $v$   | $disk$                            | DBT blending disk radius            |
| $r$                  | diameter / side length      | $th_o$                            | NeX occlusion check threshold       |
| $FoV$                | optional for forward-facing | $\alpha, d$                       | pruning parameters                  |

### 4.3 Processor

In our processing module, we first implemented an SfM interface, linked to the SfM command lines of COLMAP. After running SfM, camera poses are recovered. All the three geometric proxies our system supports require this step. As we have the ground truth of the virtual camera poses, user can optionally use the ground truth camera poses as the known poses in the COLMAP function for a calibration. We have provided a function for a conversion from left-hand z-up coordinates used by Unreal to right-hand y-up coordinates that is required by COLMAP. From our observations, the camera poses restored from the native COLMAP SfM on our synthetic dataset is worse than a restore from the real-world photographs, so a calibration is recommended. One possible reason is that even if we enabled physical-based rendering functions in Unreal, the rendering results are still quite different from real-world photographs, which may cause errors in feature detection and matching in SfM.

From here, the processor branches to get different geometric proxies. For generating explicit global 3D meshes, we use COLMAP's MVS module for stereo patch matching, dense points fusion and eventually using Delaunay triangulation to generate a 3D mesh in a .ply format. For generating implicit per-view meshes, we utilized the depth maps estimated from each camera pose in MVS.

Our implementation is straightforward: instead of using COLMAP to fusion an integrated global mesh, we do a fusion for every depth map to get local meshes. After the reconstruction, per-view meshes are stored as a set of .ply files. Besides, if the user chose to store depth maps from capturing, there is an option to optimize the depth map reconstruction in the MVS: We subtract the reconstructed depth map from the ground truth (i.e. use the ground truth value to replace the value that exceeds the error threshold). Note that we cannot directly use the ground truth depth map to do the fusion, since the depth information of translucent objects is eliminated in the game engine’s depth buffer.

For neural network models, geometry information from the checkpoints needs to be re-translated into mesh, i.e. from the 3D space occupancy probability provided in the model to a collection of vertices. NeX does not have a native support for this type of conversion. But we can find references from [18] and [21]: divide the whole 3D space into cubes, and do a cube marching that compares the occupancy probability ( $\sigma$ ) with a threshold to determine if there is a vertex at the location within the cube. After traversing the cubes in the scene, we finally get a collection of vertices as the converted mesh.

The parameters for users to experiment for the processing module are listed in Table 1.

#### 4.4 Renderer

For our rendering module, since it is running upon game engines, the implementation is based-on shaders. We have implemented three rendering shaders: ULR on explicit global mesh, DBT for per-view meshes and NeX renderer for the vertices collection. We have made VR adaptations and/or optimizations for the three methods based on their original design and implementation.

For the original ULR method, all the source images (i.e. textures) can be candidates to contribute in texturing vertices. A brute force implementation is with computation complexity of  $O(n^2)$ , which can be performance intense for complex scenes with large number of triangles. In our implementation, we design a checklist to pre-filter the vertices versus textures:

- check the boundary: check if vertices’ projection fall out of the uv boundary of the texture.
- check the occlusion: an occlusion is likely to happen when the vertices projected depth and the data from the depth map differ a lot (i.e. 20 engine units).
- check the normal: check if the vertices and texture are backward-facing.

If any of the case happens, the texture will never contribute to the vertices in the runtime rendering. The record of the pre-filtering is stored by an offline processing. With our optimization, the runtime rendering for ULR achieve better performance since traverse got canceled for unpaired vertices and textures.

At runtime, given the novel view’s transform, assign the weights to the  $n$  most relevant textures. Textures closer to the novel view’s transform will be assigned a

higher weight, since textures closer to the camera have higher resolution, and the color trace back from the ray looking at a similar direction is also similar in the common cases. (It is not necessarily applicable to complex BRDF and occlusion, which brings in a drawback for the blend-based texturing method.) The color for the vertices are the blended color from the selected textures. Note only the textures passed the above checklist contribute in blending for some vertices.

Disk-based blending is applied upon per-view meshes. The method is adapted from [19]. At runtime, per-view meshes are wrapped when overlap (transform of the vertices blend). The wrapping weight for the meshes is based on their original camera poses' relative transform to the novel view, which is logically similar to the texture weighting mechanism applied in ULR. While the per-view meshes participating in the mesh wrapping for the novel view have their corresponding textures, a disk is set for masking the textures. Only the vertices whose uv map within the disk are rendered out to save the rendering budget. All the disks add together to complete the novel view. When the radius of the disk is too small, there will be holes and seams; When the radius is too large, there will be blurry artifacts as more disks overlap.

The logic to render NeX vertices is similar to the process we applied for ULR, except for how to check an occlusion, as there is no depth map reconstructed from NeX for cameras. However, we still can find a way to check if an occlusion happens in between a pair of camera and vertex: check the  $\sigma$  function on the ray from the camera to the vertex. For all the steps along the ray, if all the steps'  $\sigma$  are lower than the threshold, we think there's no occlusion in between, on the contrary, if any of the step's  $\sigma$  is higher than the threshold, the vertex is occluded from the camera.

We adapt a pruning method to approach the target fps. The pruning works when fps drops with 2 mechanisms:

- Partial of the vertices that projected to the outer most  $\alpha$  degree of the novel view skip rendering for the current frame.
- Partial of the vertices that are far away than the threshold  $d$  from the novel view skip rendering for the current frame.

Both  $\alpha$  and  $d$  are adjustable. Using the pruning, we can handle the occasional fps drops.

As the previous modules, parameters that can be tweaked for rendering are listed in Table 1.

#### 4.5 Performance Aid

However, if the synthetic scene is too complex with a large number of triangles, the pruning method can help a little. Rendering the synthetic scene directly in VR will still be disturbed by low fps. Here, we introduce another option to further boost VR performance. We adopt a method to re-process and simplify the synthetic scene while maintaining fidelity, which is called *Seurat* [10][6].

The pipeline of Seurat goes through a headbox re-define, data re-capturing, re-processing, and rendering. Seurat uses cubemap as the datapoint. In addition to RGB, Seurat also requires the depth map, thus it is only applicable when an explicit geometric proxy (i.e. a global 3D mesh) is selected for the scene synthetic method. After the RGB-D cubemaps re-capturing, Seurat generates a layer of quad tiles that roughly represent the 3D mesh from a combination of depth maps. Along with the quad tiles, Seurat generates a corresponding texture atlas. Eventually, quad tiles and texture atlas are combined to reconstruct an simplified scene.

In the process of integrating Seurat, we re-write its out-dated capturing module with the same "High Resolution Screenshot" function that we applied in our pipeline. In this way, the adapted Seurat can provide the correct result. A comparison is illustrated in Figure 2.

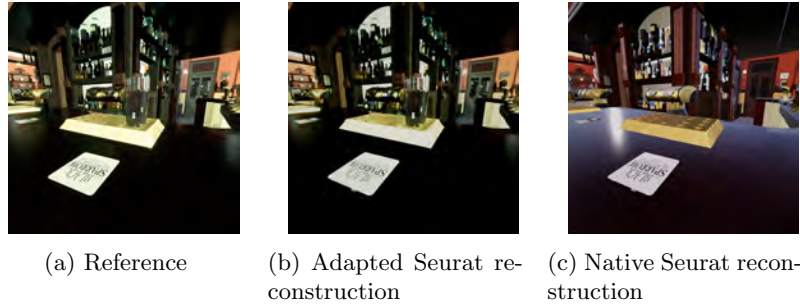


Fig. 2: Comparison of the reference test scene and the scene reconstructed by Seurat.

## 5 Experiment

After implementing all the modules, our system is ready to be tested. Here we provide an experiment towards a research question: what is the relationship of input datapoints' pattern and the perceptual quality of the synthetic views when considering the eccentricity in visual acuity in the HVM?

### 5.1 Synthetic Datapoints Setup

The synthetic scene we prepared as a test-bed completely replicates a bar in the real-world. For the purpose of investigating the raised question, we have a headbox set up in the scene, which is called "Gallery". The headbox contains diffuse surfaces with some glossy objects. As for the capturing parameters, the headbox is forward-facing sphere type, with a radius of 100 engine unit, a resolution of

$1024 \times 1024$ , and a FoV of  $120^\circ \times 180^\circ$  (H x W). (This FoV is the approximate inadvertent visual range when the head moves at will.) First we use a preliminary test to figure out a balanced datapoints density that can maintain a good perceptual quality level while saving the processing time with the target FoV. We set different level of  $v$  of the density parameter  $\rho$  in the headbox and run COLMAP to generate corresponding explicit 3D meshes, and observe if there are major seams or holes in the mesh. Then to investigate the research question, with an adequate  $v$  set for cone density, we set the pattern of datapoints on different level of  $\omega$ . We have a top-view illustration of the datapoints as shows in Figure 3:

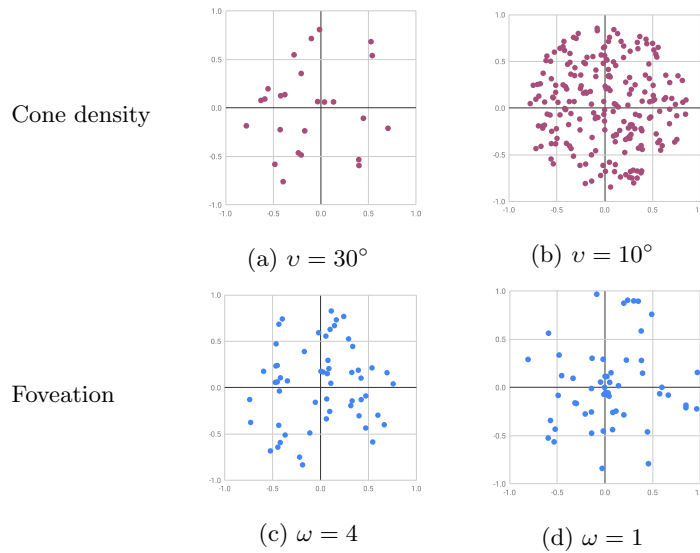


Fig. 3: Sampling density and pattern illustration.  $v$  stands for the angular cone density, and  $\omega$  stands for the level of foveation: the larger the level, the more even the distribution is.

## 5.2 Objective Metrics

For the objective metrics implemented in the system, FovVideoVDP can fit the experiment’s well best, because it is the only metric that takes into account the gaze point on the view and the HVM features such as eccentricity in visual acuity in its model. Therefore, for the evaluation of this experiment, FovVideoVDP metric is the target metric while other metrics are for the reference.

### 5.3 Results

For the preliminary test for finding an adequate  $\rho$  for a FoV of  $120^\circ \times 180^\circ$ . An visual inspection shows that at a level of  $v = 20^\circ$ , the seams and holes become minor in the reconstructed explicit 3D mesh which indicate the turning point of an adequate density.

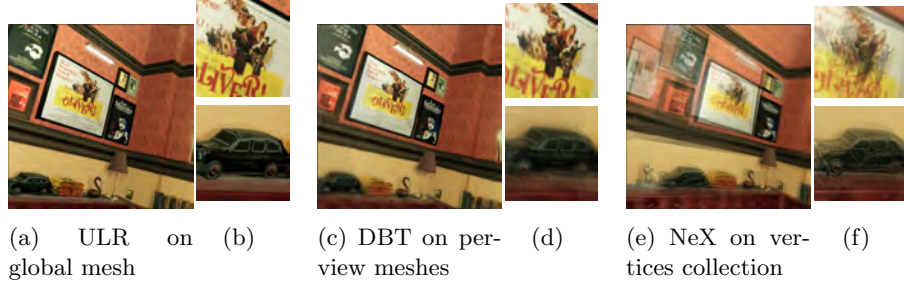


Fig. 4: A comparison of the view synthesis result on the Gallery views for the optimized ULR on global mesh method, DBT on per-view meshes and NeX on vertices.

Figure 4 illustrates the scene synthesis result from the three methods we implemented in the system (i.e. ULR on global mesh, DBT on per-view meshes, and NeX on vertices collection) for the research question in selected views from "Gallery". From a visual inspection, compared with the other 2 methods, the drawback for DBT on per-view meshes is more obvious in the mesh edges. An explanation is that when wrapping the local meshes on the occlusion edges, some errors are introduced, since the threshold and weights for wrapping are both empirical number. The robustness of the algorithm is depend on the quantity and complexity of the occlusions when we manually adjust the parameters in practice. NeX suffers from blurry artifact on the vertices collection. Increasing epochs number in training may improve the result. ULR on global mesh presents a best result under visual inspection for the tested scenario regarding to both seams & holes and blurriness.

The metrics results show in Table 2 further explain what we can observe from a visual inspection. The colors map a scale of blue-white-red for each metric, with blue meaning better quality and red worse quality. MS-SSIM shows high scores (i.e. above 0.87) for all the dataset patterns among the tested methods, which indicates a good structural similarity in general for the scene synthesis methods implemented in our system. Lower LPIPS scores indicate that the ULR on global mesh has a better perceptual quality which is important for VR applications. FoVVideoVDP and 1-NCC both give a better score for ULR on global mesh. Since these two metrics are good indicators for rephotography methods. All the metrics provide a consistent evaluation, in general, we can say ULR on global mesh provides a better reconstruction in our system for the tested scenario.

Table 2: Quality scores for the selected novel view in Gallery

|              | FovVideoVDP  | LPIPS | MS-SSIM | 1-NCC |     |
|--------------|--------------|-------|---------|-------|-----|
| $\omega = 1$ | <b>8.722</b> | 0.102 | 0.98    | 0.811 | ULR |
|              | <b>7.18</b>  | 0.257 | 0.901   | 0.798 | DBT |
|              | <b>7.01</b>  | 0.243 | 0.877   | 0.736 | NeX |
| $\omega = 2$ | <b>8.618</b> | 0.148 | 0.974   | 0.801 | ULR |
|              | <b>7.161</b> | 0.265 | 0.905   | 0.784 | DBT |
|              | <b>7.1</b>   | 0.259 | 0.883   | 0.747 | NeX |
| $\omega = 4$ | <b>8.609</b> | 0.104 | 0.973   | 0.805 | ULR |
|              | <b>7.064</b> | 0.263 | 0.899   | 0.776 | DBT |
|              | <b>6.885</b> | 0.278 | 0.874   | 0.705 | NeX |

For the evaluation towards the capturing pattern, we focus on the scores from FoVVideoVDP since that is the most relevant score for HVM-based features such as eccentricity in the visual acuity. FovVideoVDP uses  $Q_{JOD}$  (just-objectable-difference score) to quantify the quality. We asked FovVideoVDP to take the center pixel in the view as the gaze point across the evaluation. As we can see from Table 2, when the  $\omega$  is lower, the datapoints are more concentrated in the importance sampling for foveation. The corresponding score is equivalent or better than the higher  $\omega$  patterns for the method ULR on global mesh and DBT on per-view meshes. As for the exception value for  $\omega = 2$  in NeX on vertices collection, it may be related to specular datapoints' contribution, repeated experiments may average the singular value out. However, the score difference is minor, suggesting that the pattern for capturing may not contribute to the perceptual quality as much as other parameters such as density.

**Frame timing with performance aid** We also performed a frame timing test with performance aid module enabled. Result shows the re-processing can maintain a good fidelity within the headbox (Figure 2). The original scene has 1.3 m triangles in total. With the ray-tracing refraction enabled for high-definition rendering, the average rendering latency on a high-end computer (with Nvidia GTX 2080 Ti) is around 43 ms. In the reconstructed scene from Seurat, the total triangle number downsized to 72k, and we got an average latency of 5.9 ms in VR on the same PC.

## 6 Discussion

At present, modules in our system are applied in the form of actor blueprints in Unreal. We have realized that the system needs further integration - better to



be in the form of a integrated plug-in for the game engine. As Dupont et al. [3] did for Unity.

We have obtained more robust geometric proxies through the calibration in our capturing and processing modules. A manual calibration in the dataset capturing to filter out the datapoints with features less than 200 from entering into the following pipeline is suggested. By experiments we found images with less than 200 feature points sometimes cause an error in the following mesh reconstruction with wrong mapping among other features.

For NeX modelling, in its original implementation, the training time for a 17 images dataset with a 4000 high epoch runs for 18 hours [29]. For the practical reason, we had to decrease the training epoches. Coupled with the information loss in the adaptive conversion to integrate the model to Unreal, the quality scores in our system for NeX are lower than the scores they reported.

As for the evaluation module, it can be further tailored for VR. Since fps can determine a comfortable VR experience, integrate a temporal metric has the potential to support spatial-temporal quality evaluation for VR.

If we push our system to dynamic scenes, much interesting content is coming along. However, it requires geometric proxies to be temporal consistency. Also there will be a much stricter requirement for the accuracy as human sensitivity to perceptual effects over time is high, e.g., any flickering at the edges of objects on a geometry reconstruction is particularly noticeable.

## 7 Conclusion

In this paper we presented a system that used four modules to integrate a whole pipeline of scene synthesis for VR. The capturing module was calibrated for better pose restoration, which supported both real-world photographs and synthetic images. The processing and rendering module adapted and/or optimized three representative scene synthesis methods with various geometric proxies, in which the neural-network-based method NeX became available for VR with our adaptation. The evaluation module was VR-tailored, based on HVM, perceptual quality, structural similarity, and rephotography. In addition, we provided an optional performance-aid module for performance-intense use cases. The proposed system can be widely applied in scene synthesis for VR and support further research.

We have proved the practicability of the system by demonstrating the pipeline through an experiment studying which of the tested capturing strategies performs best. Experimental metrics evaluations indicate that the foveation capturing pattern performs best when applied on the tested scenario.

## References

1. Bertel, T., Campbell, N.D.F., Richardt, C.: MegaParallax: Casual 360° panoramas with motion parallax **25**(5), 1828–1835. <https://doi.org/10.1109/TVCG.2019.2898799>, conference Name: IEEE Transactions on Visualization and Computer Graphics

2. Buehler, C., Bosse, M., McMillan, L., Gortler, S., Cohen, M.: Unstructured Lumigraph Rendering. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. pp. 425–432. SIGGRAPH '01, ACM, New York, NY, USA (2001). <https://doi.org/10.1145/383259.383309>, <http://doi.acm.org/10.1145/383259.383309>
3. Dupont De Dinechin, G., Paljic, A.: From Real to Virtual: An Image-Based Rendering Toolkit to Help Bring the World Around Us Into Virtual Reality. In: 2020 IEEE 6th Workshop on Everyday Virtual Reality (WEVR). Atlanta, Georgia, United States (Mar 2020), <https://hal.archives-ouvertes.fr/hal-02492896>
4. Flynn, J., Broxton, M., Debevec, P., DuVall, M., Fyffe, G., Overbeck, R., Snavely, N., Tucker, R.: DeepView: View Synthesis With Learned Gradient Descent. pp. 2367–2376 (2019)
5. Foster, S., Halfstein, D.: Integrating 3D modeling, photogrammetry and design. Springer (2014)
6. googlevr: Seurat (2019), <https://github.com/googlevr/seurat>
7. Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M.F.: The lumigraph. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. pp. 43–54. SIGGRAPH '96, Association for Computing Machinery. <https://doi.org/10.1145/237170.237200>, <https://doi.org/10.1145/237170.237200>
8. Hedman, P., Ritschel, T., Drettakis, G., Brostow, G.: Scalable inside-out image-based rendering **35**(6), 231:1–231:11. <https://doi.org/10.1145/2980179.2982420>, <https://doi.org/10.1145/2980179.2982420>
9. Jonathan: Are there high-poly source meshes? (2018), <https://help.quixel.com/hc/en-us/articles/115000604189-Are-there-high-poly-source-meshes->
10. Lall, P., Borac, S., Richardson, D., Pharr, M., Ernst, M.: View-Region Optimized Image-Based Scene Simplification. Proc. ACM Comput. Graph. Interact. Tech. **1**(2), 26:1–26:22 (Aug 2018). <https://doi.org/10.1145/3233311>, <http://doi.acm.org/10.1145/3233311>
11. Levoy, M., Hanrahan, P.: Light field rendering. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. pp. 31–42. SIGGRAPH '96, ACM. <https://doi.org/10.1145/237170.237199>, <http://doi.acm.org/10.1145/237170.237199>
12. Lin, W., Jay Kuo, C.C.: Perceptual visual quality metrics: A survey **22**(4), 297–312. <https://doi.org/10.1016/j.jvcir.2011.01.005>, <http://www.sciencedirect.com/science/article/pii/S1047320311000204>
13. Liu, L., Gu, J., Zaw Lin, K., Chua, T.S., Theobalt, C.: Neural sparse voxel fields. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 15651–15663. Curran Associates, Inc. (2020), <https://proceedings.neurips.cc/paper/2020/file/b4b758962f17808746e9bb832a6fa4b8-Paper.pdf>
14. Luo, B., Xu, F., Richardt, C., Yong, J.: Parallax360: Stereoscopic 360° scene representation for head-motion parallax **24**(4), 1545–1553. <https://doi.org/10.1109/TVCG.2018.2794071>, conference Name: IEEE Transactions on Visualization and Computer Graphics
15. Mantiuk, R.K., Denes, G., Chapiro, A., Kaplanyan, A., Rufo, G., Bachy, R., Lian, T., Patney, A.: Fovvideovdp: A visible difference predictor for wide field-of-view video—supplementary material (2021)
16. Meng, X., Du, R., Zwicker, M., Varshney, A.: Kernel foveated rendering. Proceedings of the ACM on Computer Graphics and Interactive Techniques **1**(1), 1–20 (2018)

17. Mildenhall, B., Srinivasan, P.P., Ortiz-Cayon, R., Kalantari, N.K., Ramamoorthi, R., Ng, R., Kar, A.: Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)* (2019)
18. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: *ECCV* (2020)
19. Overbeck, R.S., Erickson, D., Evangelakos, D., Pharr, M., Debevec, P.: A System for Acquiring, Processing, and Rendering Panoramic Light Field Stills for Virtual Reality. *ACM Trans. Graph.* **37**(6), 197:1–197:15 (Dec 2018). <https://doi.org/10.1145/3272127.3275031>, <http://doi.acm.org/10.1145/3272127.3275031>
20. Pharr, M., Jakob, W., Humphreys, G.: *Physically based rendering: From theory to implementation*. Morgan Kaufmann (2016)
21. Quei-An, C.: Nerf\_pl: a pytorch-lightning implementation of nerf (2020), [https://github.com/kwea123/nerf\\_pl/](https://github.com/kwea123/nerf_pl/)
22. Richardt, C., Tompkin, J., Wetzstein, G.: Capture, reconstruction, and representation of the visual real world for virtual reality. In: Magnor, M., Sorkine-Hornung, A. (eds.) *Real VR – Immersive Digital Reality: How to Import the Real World into Head-Mounted Immersive Displays*, pp. 3–32. *Lecture Notes in Computer Science*, Springer International Publishing. [https://doi.org/10.1007/978-3-030-41816-8\\_1](https://doi.org/10.1007/978-3-030-41816-8_1), [https://doi.org/10.1007/978-3-030-41816-8\\_1](https://doi.org/10.1007/978-3-030-41816-8_1)
23. Schönberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: *Conference on Computer Vision and Pattern Recognition (CVPR)* (2016)
24. Schönberger, J.L., Zheng, E., Pollefeys, M., Frahm, J.M.: Pixelwise view selection for unstructured multi-view stereo. In: *European Conference on Computer Vision (ECCV)* (2016)
25. Shum, H.Y., Chan, S.C., Kang, S.B.: *Image-Based Rendering*. Springer Science & Business Media, google-Books-ID: 93J3Z96ZVwoC
26. Waechter, M., Beljan, M., Fuhrmann, S., Moehrle, N., Kopf, J., Goesele, M.: Virtual rephotography: Novel view prediction error for 3d reconstruction **36**(1), 8:1–8:11. <https://doi.org/10.1145/2999533>, <http://doi.org/10.1145/2999533>
27. Waechter, M., Moehrle, N., Goesele, M.: Let there be color! large-scale texturing of 3d reconstructions. In: *European conference on computer vision*. pp. 836–850. Springer (2014)
28. Wang, Z., Li, Q.: Information Content Weighting for Perceptual Image Quality Assessment. *IEEE Transactions on Image Processing* **20**(5), 1185–1198 (May 2011). <https://doi.org/10.1109/TIP.2010.2092435>, conference Name: *IEEE Transactions on Image Processing*
29. Wizadwongsa, S., Phongthawee, P., Yenphraphai, J., Suwajanakorn, S.: Nex: Real-time view synthesis with neural basis expansion. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2021)
30. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. pp. 586–595
31. Zhou, T., Tucker, R., Flynn, J., Fyffe, G., Snavely, N.: Stereo magnification: learning view synthesis using multiplane images **37**(4), 65:1–65:12. <https://doi.org/10.1145/3197517.3201323>, <https://doi.org/10.1145/3197517.3201323>