WinSyn: Appendix

1. Overview of the Supplemental

This supplemental document complements our paper titled "WinSyn: A High Resolution Testbed for Synthetic Data." It expands on our work by providing in-depth details and additional data supporting our research into high-quality synthetic data generation through procedural modeling techniques. Within this document, we offer a comprehensive look into the diversity of the photos used in our dataset, including their geographic origins and the number of images processed. We also present visual examples of the dataset, including photographs, labels, synthetic images, and RAW images, alongside an illustration of the distribution of crop sizes.

Further, we delve into the specifics of the labels used to annotate our images, including the guidelines provided to labelers, ensuring clarity and consistency across the dataset. The document includes a further examination (beyond what was covered in the main paper) of the experiments conducted to understand the impact of mixing real and synthetic data on semantic segmentation tasks.

The details of our procedural model are explained. We explore various modifications to the model, such as adjustments in the number of rendering samples, wall materials, mixed materials, lighting models, camera positions, window geometry, and the subset of labels captured. These variations were evaluated to enhance the procedural model's fidelity and its application in replicating spatial semantics of real scenes.

2. Photo Diversity

The numbers of photos, raw images, and their geographic origin are shown in Table 1 on page 2. The dataset covers 28 countries, with notable contributions from Austria, the USA, and Germany in terms of both raw and labeled images. This diversity is important for our goal of capturing a wide range of architectural styles, influenced by culture, climates, and historical contexts, which influence window designs and materials. The total collection includes 75,739 photos, with 69,713 RAW images (6,666 of them annotated), and 89,318 distinct windows (9,002 annotated).

3. Dataset examples

We provide examples of the photographs, labels, synthetic images, and RAW images in Fig. 3 on page 4, Fig. 4 on page 5, Fig. 6 on page 7, and Fig. 5 on page 6, respectively. Fig. 2 on page 3 illustrates the distribution of crop sizes.

4. Labels

Figure 1 shows the fraction by pixels of the labeled data. We use the following labels when labeling the real and synthetic data:

1. **window-pane**: Glass, painted "glass", opening (missing glass), mesh-screen, or repair (e.g., wood/brick covering a broken pane)
2. **window-frame**: Not part of the wall, part of the window, usually wood, metal, or plastic. Used infrequently for door frames.
3. **open-window**: The interior of the building.
4. **wall-frame**: Part of the wall which is adapted to the window. Each of the following should have this label, but be different instances:
   (a) *window apron* (sill; part of the wall, below the window)
   (b) *window header* (lintel; part of the wall, above the window)
   (c) *wall frame* (part of the wall; decorates/supports the window)
   (d) *balcony base* (support; below balcony, holds up the balcony railings)
5. **wall**: Other parts of the wall
6. **shutter**: These are beside the window and swing sideways to protect and insulate the window.
7. **blind**: Exterior blinds. these are above the window and move down to protect and insulate the window.
8. **bars**: Fixtures protecting the window and frame.
9. **balcony**: Guard-rails, railings, balconette.
10. **misc-object**: In front of the glass: people, pot-plants, toys, junk, pipes, wires, trees, plants on the wall, alarm, unknown objects, misc. foreground.

Other guidance provided to labelers:

- Objects seen through or "inside the frame" of the main windows should be ignored. For example, if you can see another window through the main window then ignore it.
- Objects reflected (in the glass) should be ignored.
- Fine window structures (leaded glass, "fake" (plastic) leaded glass inside glass, security chain link fence, chicken wire) should be ignored. Larger structures (cast

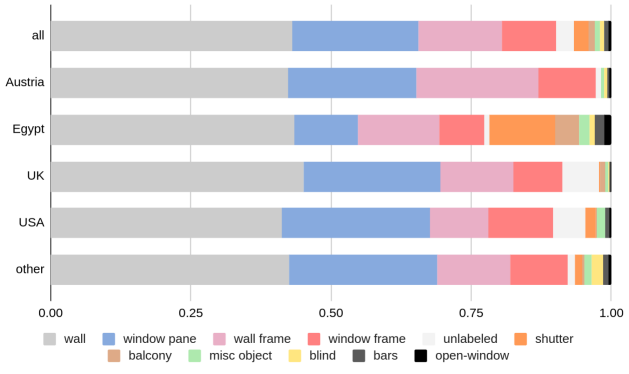


Figure 1. The label fraction for the different geographic partitions of the labeled data. These statistics were collected over square sub-cropped label masks.

Location	Images	Raw	Labeled	Crops	Labeled Raw
Algeria	1714	1714	0	1795	0
Argentina	4153	4153	243	4610	243
Armenia	2135	2135	0	2143	0
Austria	9494	9494	2139	13966	2139
Bangladesh	1996	1996	64	2224	64
Brazil	2102	2102	35	2750	35
Canada	1999	1999	0	2115	0
China	2115	2114	0	2466	0
Columbia	663	663	0	685	0
Cyprus	2077	2077	93	2070	93
Czechia	2183	2183	102	2159	102
Denmark	233	0	78	254	0
Egypt	3873	3855	1500	4436	1496
Germany	4136	4136	861	6083	861
Greece	2045	2045	62	2085	62
India	1943	1934	0	2143	0
Ireland	2057	2057	58	2049	58
Macedonia	2024	2024	79	2050	79
misc	184	0	32	171	0
Morocco	2006	2006	0	2013	0
Philippines	1829	1828	0	2170	0
Poland	4072	4072	20	5851	20
Saudi Arabia	213	213	0	238	0
Tanzania	1597	1597	0	1644	0
Thailand	1009	1009	44	1052	44
Turkey	3027	3025	0	3116	0
UK	5541	481	2015	6760	0
USA	9319	8801	1577	10220	1370
Totals	75,739	69,713	9,002	89,318	6,666

Table 1. Breakdown of the number of photographic images by location, along with the number of images with labels, the number in RAW format, and the number of crops (rectangle around individual windows). Note that more than one window can be cropped from a photograph.

iron fence) should be labeled. If uncertain, ignore.
- Label other objects in front of the window as misc-object. Other objects (e.g., in front of the wall) may be left unlabeled or also labeled as misc-object.

5. Dataset and Architecture Experiments

We performed two experiments to study the impact of different mixes of real and synthetic data on the segmentation task. For all experiments, we report the all-class mIoU excluding the 'unlabeled' label. The test size was 4,906 real or synthetic data. The first experiment is presented in Figure 7; it shows the performance when training on different amounts of real or synthetic data; we test on both real and synthetic data. The second experiment, in Figure 8, illustrates the impact of different mixtures of real and synthetic data, when tested on real data.

The segmentation task is highly sensitive to the network and data selection. Therefore, we examine several approaches to cross-domain learning; none of them are able to close the real/synthetic performance gap. This suggests that graphical and modeling improvements to the procedural model remain important. Table 2 introduces a number of different approaches and their task performance. We choose not to focus on specialist domain transfer (Label Adaptation or Unsupervised Domain Adaptation) approaches because of the blurring of real and synthetic data during training and the increased complexity of the techniques.

- DeepLabV3+ [4] is a more traditional deep convolutional neural network which is used here without pre-training. Figure 9 explores the impact of using this network for the segmentation task, showing similar patterns to BEiT (cf. Figure 7).
- BEiTv2 [14] is a modern transformer-based foundation architecture that is pre-trained on other datasets. It has a 'base' model with 86M parameters and a 'large' model

with 300M parameters. The 'base' network typically trains in 7 hours on an NVIDIA V100 card. The decision to use BEiTv2 'base' for the main experiments and variations in this paper provides a realistically challenging task balancing a reasonably quick evaluation with an architecture representative of modern machine learning.

- Histogram Matching [7] aligns the training data's per-channel brightness distribution with that of the real-world training data. Unlike the unsupervised per-image histogram equalization (see Appendix section 7.4), this is semi-supervised. The method led to a minor mIoU improvement, from 32.58 to 32.96.
- Label Adaptation is a technique explored by [15] for enhancing the utility of synthetic data. Although this technique does not closely align with our goal of minimizing labeled image use, it yielded a significant improvement in mIoU, from 32.58 to 41.55. This improvement is comparable to the effect of adding 64-128 real samples to our synthetic training set. However, we contend that label adaptation may present an overly optimistic view of the efficacy of synthetic data, as it often leverages real data labels as a shape prior. In our results, we train the label adaption network with BEiT and 4,096 real labels. As Figure 10, we observe that improving labeling by just learning from the labels provides a powerful supervised prior to improve the accuracy between dataset. This semi-supervised technique is able to identify common problems with the synthetic-trained labeling network and correct them.
- Masked Image Consistency (MIC) [10] is an Unsupervised Domain Adaptation technique that uses unlabeled real-world data to improve the training results. It is a transformer-based architecture with several recent improvements including Rare Class Sampling and an ImageNet Feature Distance [9].
- We also provide results for an 'easy' real data partition (in contrast to the standard 'all' partition used for other experiments) which might a stepping-stone when creating procedural models for challenges. We created a subset of labeled photos that contained only rectangular windows, without significant additional complexities (such as bars, blinds, occlusions, or other miscellaneous objects). To limit the modeling complexity, we evaluate this partition with 3 labels (window pane, window-frame, and wall); all other labels are ignored. The easy partition is of size 4,113, divided into 2,048 training and 2,065 test images. The easy test partition is disjoint to the standard test partition used in our other experiments. We experiment with training size in Figure 9.

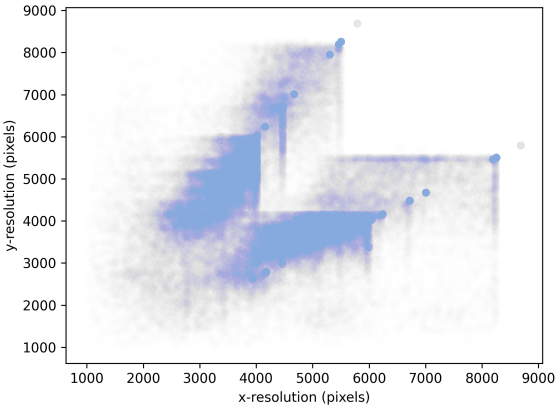


Figure 2. A scatter plot of the resolution of the 89,318 crops taken from these photos.

6. Procedural Model Details

This section provides additional details about the *baseline* procedural model used to create the scenes. Figure 11 provides an overview.

Building mass. The walls, roof, bay windows, and wings are created using the *Computer Generated Architecture* (CGA) procedural modeling language [13]. The grammar we use creates a rectangle on the floor-plane between 3 and 9 meters wide and 3 and 5 meters deep. This is extruded upwards to create between 1 and 4 stories. Wings are optionally created from the side faces; bay-windows are extruded on the front of the mode. Hip, shed, or gable roofs are added to the top of these masses. The front of the building and bay walls are split to create walls and window rectangles. Optionally, a timber-frame is created within wall panels.

Window geometry is created within a rectangle proscribed by the CGA building grammar. Because CGA isn't able to model curves or extrude profiles, we use a second split grammar to create a variety of window shapes. The output of the grammar is a grouped hierarchy of Bézier splines, each assigned a profile, which is applied to create a frame around a pane of glass. This allows the creation of complex windows; for example, the outermost group may be a window casing (around the edge of the window), a second group has a different profile to model static panels, while a final group of opening panels has a third profile. A single set of profiles is selected from 14 sets available (Figure 21, right). Each spline is then assigned one profile within the set and filled with glass. The split grammar sequences rules to convert the input rectangle into groups within the hierarchy.

We have four classes of shapes (rectangular, trapezoid, circular, arched). Each of these classes has several subvariants. For example, circular windows may become semi-circular or window arches may be straight or curved. Each shape has splits appropriate to that geometry (e.g., circu-

Figure 3. A random selection of our 75,739 window photos and their 89,318 crops (red boxes) from Austria, Egypt, UK, USA, and Other. Each partition contains at least 1,500 photographs. Zoom-in for easier viewing.

Figure 4. A random selection our 9,002 labeled cropped photos. Zoom-in for easier viewing.

Figure 5. Random samples from the 21,290 in the synthetic window dataset showing color and label channels. Zoom-in for easier viewing.

lar windows only split into quarters) that are applied with certain probabilities defined by the grammar. Some shapes split into each other; for example, the bottom of an arched window is a rectangle and shares the splits available.

To model open windows, parts of this hierarchy may be translated (for sash or sliding windows) or rotated (for hinged windows). These can be identified in Figure 5 by looking for the black *open-window* labels.

Figure 6. 6,666 of our photos contain the original RAW data as well as labels. Above: the default JPG image and labels (column 1), reprocessed RAW files for global brightness (2), reprocessed RAW files for a given region's (blue boxes) brightness (column 3), and a crop of the result showing the full resolution (4).

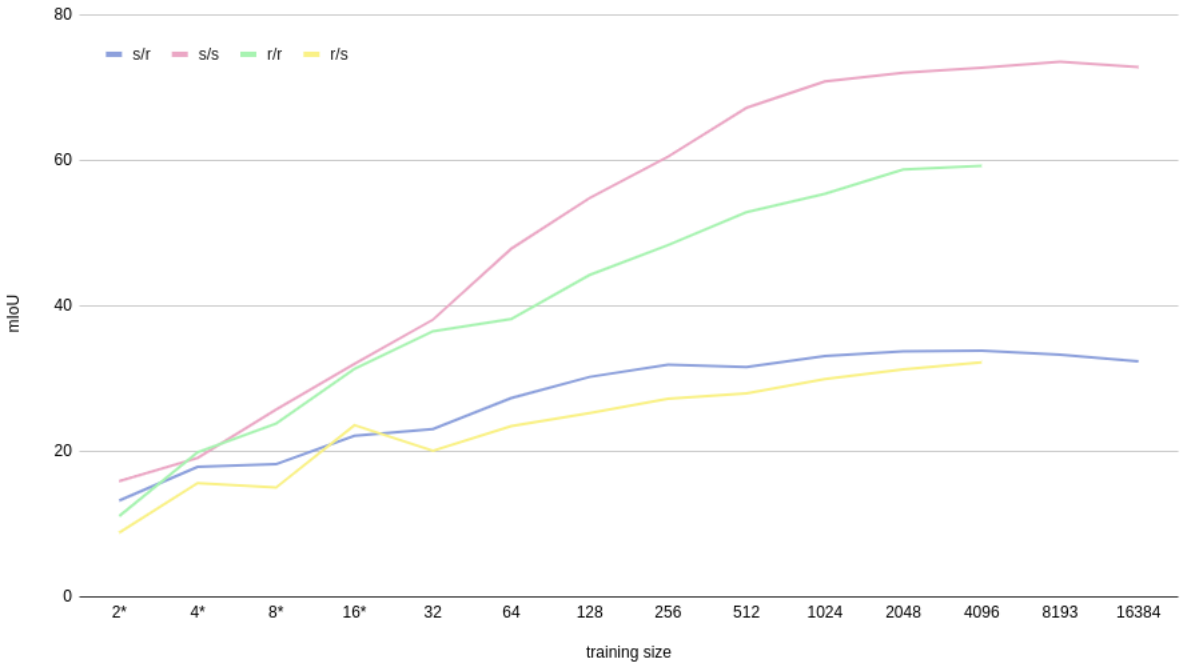


Figure 7. Performance with different amounts of synthetic and real data. e.g., s/r was trained on synthetic (s) and tested on real (r). Models with a sample size below 16 were repeated 5 times with different samples from the test set; these are marked with an asterisk.

Window surrounds (sills, lintels, complete frames) are constructed from extruded profiles over splines. These decorate the wall and recessed area around the window. We use sections of the window-shape to describe the spline shapes; for example, a window lintel is the top section of a window-shape, which may be straight, arched, or circular. The spline for a lintel or sill is optionally extruded sideways, beyond the edge of the window.

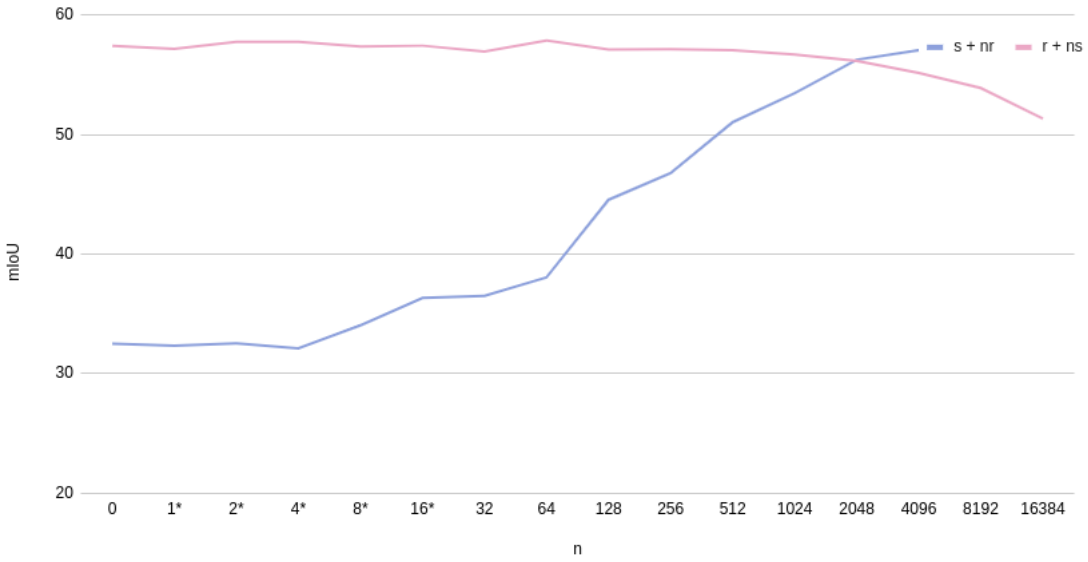


Figure 8. Segmentation mIoU on real data when training on various mixes of real and synthetic data. The blue series shows the effect of adding n real data to 2,048 synthetic images. The red series shows the effect of adding n synthetic data to 2,048 real images. Models with a sample size below 16 were repeated 5 times with a different selection of n; these are marked with an asterisk.

test partition:test size	network	train data type	train data size	mIoU
all: 4906	DeepLabV3+ [4]	synthetic	4096	20.92
	DeepLabV3+	real	4096	34.46
	BEiTv2_base [14]	synthetic	16384	32.31
	BEiTv2_large	synthetic	16384	36.44
	BEiT2_base	real	4096	59.19
	BEiTv2_large	real	4096	65.45
	BEiT2_base	Histogram Matching [7]	4096	32.96
	Label Adaptation [15]	synthetic & real labels	2048	41.55
	MIC [10]	synthetic & real unlabeled	16384	44.30
easy: 2065	BEiTv2_base	synthetic	2048	76.64
	BEiTv2_base	real-easy	2048	87.75

Table 2. An exploration of different networks, training techniques, and dataset partitions. Label adaptation uses BEiT2_regular for both sub-networks. The 'easy' partition of the real data uses three labels (wall, window-pane, and window frame) for both test and train; the 'all' partition is our standard one. All networks were evaluated on real data and used a resolution 512x512 pixels for training and testing.

There is a small probability (0.013) of creating *blind* windows (windows without a frame or glass) or a window without glass. These features were added after observing them in the dataset.

Interiors. Window-dressing is positioned inside the windows, and models curtains, Venetian blinds, and wooden or fabric blinds. The soft dressings use a cloth simulation to create bunches and gather the material for different positions and window shapes. Behind the dressing we use an 'interior-box' onto which we project a randomly selected interior panorama. At night, this box emits light, simulating a lit interior. Window dressing is only applied to the 'primary window' - the one which the camera is pointing at.

Materials. With the exception of sky-boxes, interiors, and street clutter, all geometry is textured using procedural shaders which define parameters for principled BSDF materials [2] and displacement. This gives the effect of materials such as wood, brick, stone, concrete, metal, and glass. For example, our building's walls are textured using one of three shaders — brick, wood-planks, or (optionally peeling) stucco, each controlled by between 20 and 30 parameters. For stucco, these parameters control the color and glossiness of the paint, color and texture of any underlying concrete, wear patterns between concrete and stucco, and the amount of dirt on the surface. In addition to the parameters,

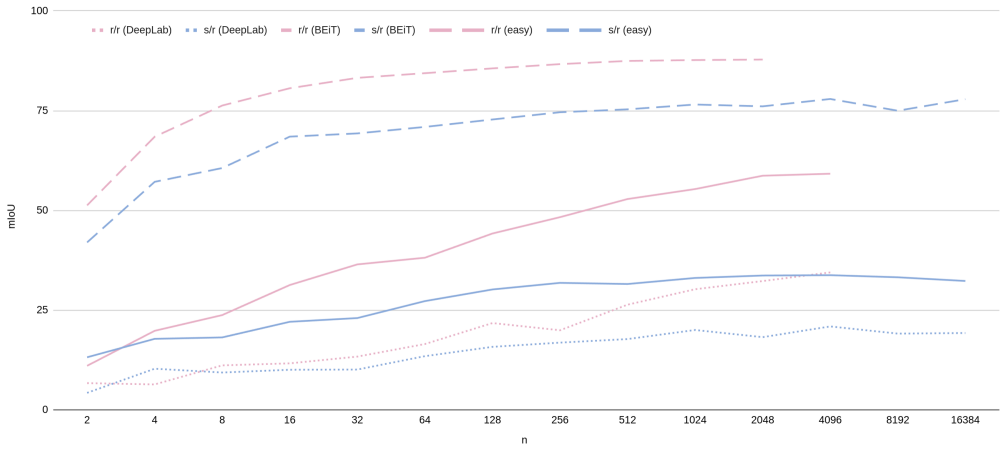


Figure 9. Segmentation mIoU performance against training size in a variety of scenarios. We show our standard BEiTv2 results (solid lines; as Figure 7; pre-trained) on the complete ('all') data partition. We compare to DeepLabv3+ [4] (dotted lines; no pre-training, 'all' partition), and BEiTv2 on the 'easy' partition (dashed lines, train and test on 'easy'). Neither improvements in modern network performance, nor using a easier simplified segmentation task, lead to a closing of the synthetic-real gap.

the input to the materials includes values from the geometry such as surface normal (more dirt gathers on horizontal surfaces), global illumination maps (deep crevices are unlikely to be damaged by wear), and 'pointiness' (corners and edges are more likely to be damaged).

Using a single material for all objects in a scene would be monotonous (but occasionally observed in the real world, where a single type of paint has been applied to walls, frames, pipes, and wall-frames), while using many materials can be unrealistic. To balance these issues, a number of techniques are used share materials between objects in the scene. For example, the wall-rectangles output from the CGA are named for their location within the grammar tree. Then, with a $\frac{2}{3}$ chance a single material is used for all wall rectangles; the remainder of the rectangles are assigned from a pool of materials of randomly selected size between 2 and the number of names. This creates the possibility of a variety of patterns of wall materials over the facade. Another example is window-frame and window-surround (sills, lintels) materials. Here we use the parameter system to share materials between different window-frames (each scene selects this per-window probability uniformly between 0 and 1), and to occasionally (with a $\frac{1}{3}$ chance) use the same material for the window-surround.

Lighting. The scene has three light sources - a panorama providing omni-directional light based on the selected skydome background image, a sun adding a directional strong light-source, and an (optionally) lit interior material allowing interior lighting at night. The sun-lamp's light filters through an 'urban canyon' that simulates the light passing around other buildings, as illustrated in Fig 11. This is a stochastically generated area of cuboids of mixed size and height behind the camera. The geometry allows indirect light paths and shadows to fall onto the building, adding image features similar to those to in the ground truth such as shadows from trees, telegraph poles, and light reflecting from other buildings. The urban canyon is not directly visible to the camera or in reflections.

The sun is rotated with an azimuth between -90 to 90 degrees, with an increased chance of being near either extreme. This increases the chance of the sun direction being parallel to the wall and creating large glancing shadows. The altitude of the sun is normally distributed with $\mu = 40, \sigma = 28$ degrees. The size of the sun also varies - creating the effect of light through different sky conditions.

With a probability of 0.05, we create a night scene. The light emission from the exterior skydome and sun is reduced, and the interior-box is lit.

Creating well-exposed images is challenging. The light, material, and geometry must be tuned to create both realistic and balanced exposures. The variety of parameters combined with the physically based renderer leads to a wide variety of exposure between very dark and very bright images. In section 7, we explore a variant of the model which adjusts the exposure dynamically.

Exterior. Buildings do not appear in isolation – to increase variety and realism we add clutter to the outside of the building. This includes objects as varied as trash cans, signs, scooters, and delivery lockers. We collected a set of 278 varied meshes and textures using a hand-held LiDAR and RGB scanner (see Fig 12) which is augmented by a set of images of 5,630 street signs [3] with the backgrounds removed. A subset of the meshes were gathered into collections which could be repeated to model repeating features

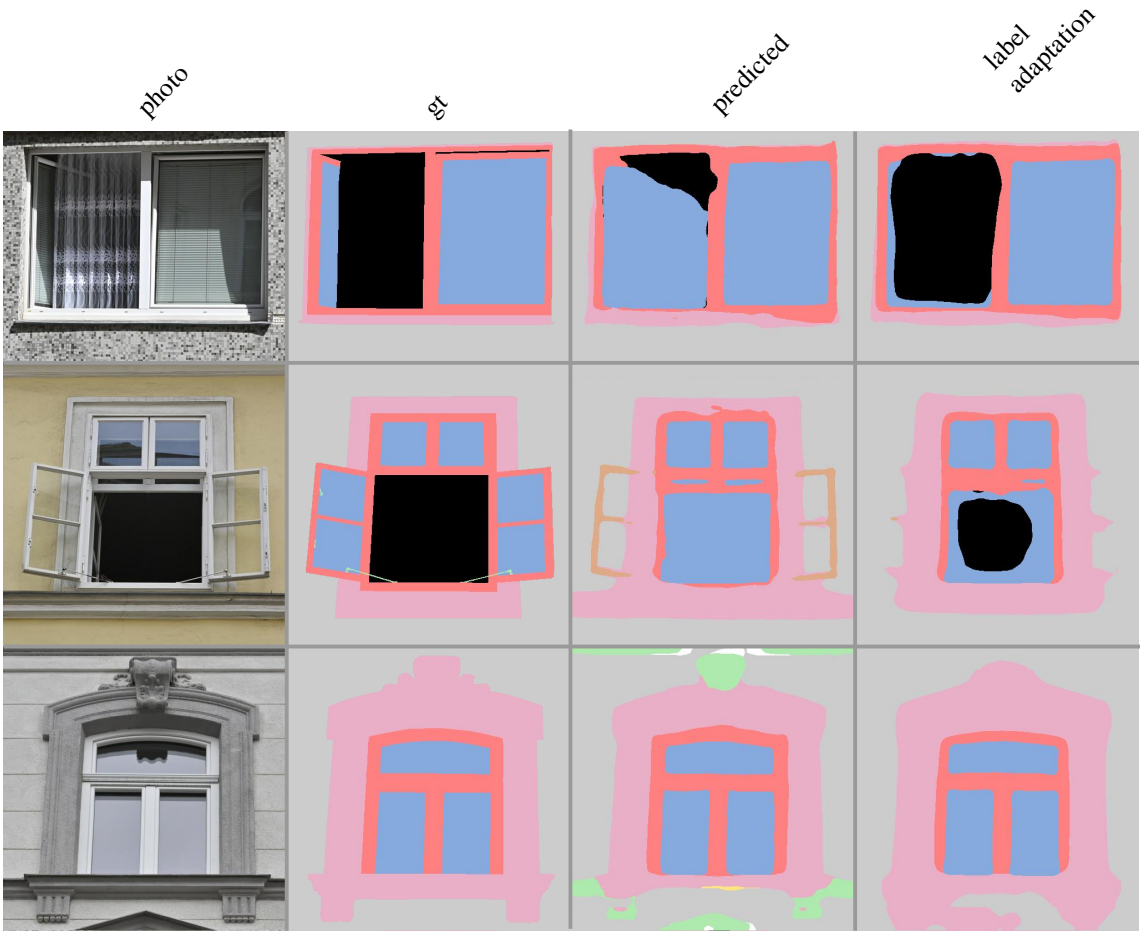


Figure 10. Example outputs from our label adaptation [15] label-to-label network. We note that the network is able to identify likely mislabeling (such as an open window without the *open-window* label) and correct the output appropriately.

such as traffic cones, rows of trees, or bollards. The primary requirements for the mesh and image clutter is that they belong in the street scene, have no personal information (e.g., car number plates) and do not contain building windows (which would degrade labeling accuracy).

Drain pipes and wires are both generated from a graph of potential edges (PEs). Paths through this graph are extruded to create drainpipe or electrical wire geometry. Using a modification to CGA – allowing a face to be split to its constituent edges – we build a graph during the evaluation of the building mass grammar. The graph edges include the gutter, and bottom of the walls, as well as vertical and horizontal edges within each rectangle in the wall. We select sources and sinks from the graph vertices (e.g., between the gutter and floor level or between two random points in the graph) and find the shortest path between them. These paths are smoothed appropriately (e.g, wires are curved and have much more variability), offset from the wall, meshes are generated by extruding profiles, and are finally decorated with textures.

A panoramic skydome adds a background that is often visible in reflections in a window. This background is se-

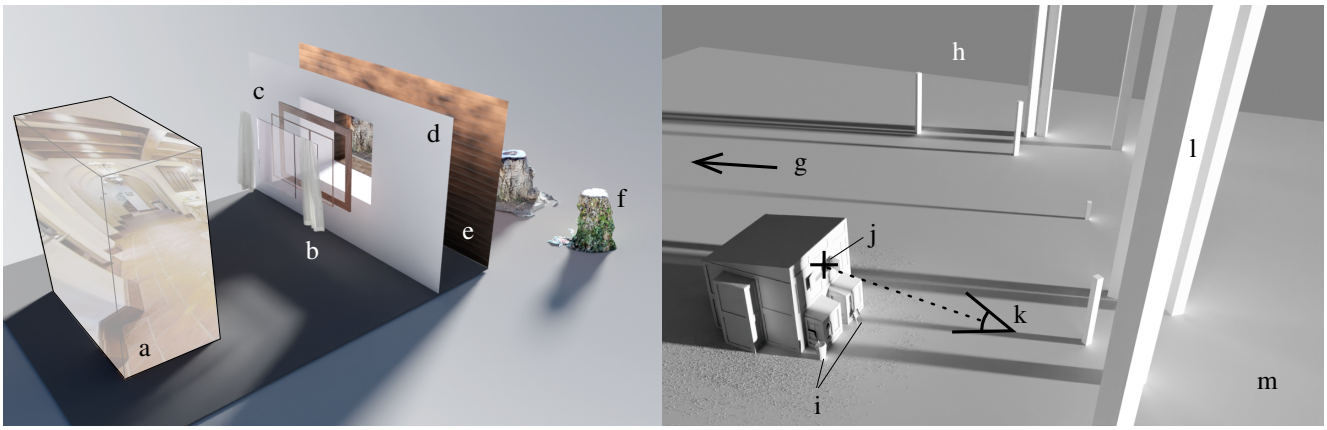


Figure 11. Left: A layered approach to window realism. The window is composed of nested layers of frame and glass (c) and is optionally dressed (b, curtains). Interior (d) and exterior (e) walls divide the inside from the outside. An interior-box (a) is the geometry onto which we project an interior panorama, while the exterior is decorated with street clutter (f). Right: Our environment rig. The building geometry lies on a floor plane (m) under a skybox (h). It is lit by the skybox and from a directional sun lamp (g), which may cast shadows from the urban canyon (l). The camera is positioned in front of the building (k), pointing towards the camera target (j) within the primary window.

lected from a library of street-view images. The background may contain other buildings' windows; to avoid degrading the quality of labeling results, this area is unlabeled. A simple circular floor supports the building.

Camera Position. The baseline camera positioning switches between two modes: one third of scenes sample the camera's position uniformly from inside a box the width of the building, between 0.5 and 1m from the ground, and 2 to 8m from the facade. The other third are sampled from a position in front of the window - sampled from a 4×2 meter box parallel to the wall positioned directly in front of the window. This combination was motivated to approximate a held camera, and occasional use of a higher camera (on a hill or from a neighboring building).

The camera's field of view is computed from the angle between the corners of the window and the camera position. It is perturbed from the largest apparent angle between window diagonals by a normally distributed factor with $\mu = 1.1\sigma = 0.1$. The camera's direction is computed with z-up, pointing towards a target uniformly sampled from a 20×20cm rectangle in the center of the window.

Occasionally a camera positioned in this way will be behind a large street clutter mesh. If 2 out of 5 key points (corners and center) on the window are occluded, the offending clutter object(s) are removed.

Parameters The design of the parameter system in a synthetic model has a number of goals. Primarily, it should create distributions of parameters with a good evaluated task accuracy and visual realism.

To achieve these goals within a complex realistic model, it is necessary to start with an estimated distribution, which can be iteratively refined with a generate-inspect-update process. While some progress has been made on automated parameter selection [12], our large number of parameters (up to 21,735 observed) and non-differentiable renderer makes this challenging; these constraints are consequences of our decision to focus on variety (a large number of parameters are required to drive and coordinate a large number of features with high variety) and realism (we use a physically based path-tracing render). Therefore, an initial parameter distribution is usually estimated by the engineer or artist who creates the synthetic model. Often these parameters will be estimated on a prototype or incomplete model. A parameter system must therefore support iterative development by tracking parameter and code changes together with their impact on task performance, and be able to explore sequences of changes in the distributions.

The parameter system should be reproducible; that is, we must be able to run the same software code-path multiple times when debugging or rendering different variations of the same geometry on a cluster of computers. It should also be robust – small changes in the model's code should have small changes in the output. These requirements together imply that the parameter system has to run in a hybrid mode, with zero or more parameters specified, and others drawn from the distribution.

The system must support both continuous (e.g., window width, wall texture color) and discrete (e.g., shape of windows, type of window dressing) parameters. The distributions WinSyn uses are typically uniform or Gaussian for continuous parameters and Bernoulli for discrete parameters. But combinations of these quickly become complex when repeated parameter selection moves control to different code branches; these branches may be a function of both parameters and geometry. An example is window-pane

splitting – we may continue to split windows until we hit a geometry constraint (they become small) or a parameter decides we should stop (e.g., maximum split depth or stopping early to create more large windows). Another example is that our building modeling language, CGA [13] uses repeated 'relative splits' in which a parameter (for example a uniform continuous window panel width parameter) is adjusted to fit an integer number of windows into a wall (a geometric constraint).

We found that apparent realism was enhanced when it was possible to share parameters between disparate parts of the model. For example, to use the same material on a wall as window-frame, balcony base, or drain pipe.

To implement these requirements WinSyn uses several mechanisms in concert to build a parameter system.

- The distributions are defined in the code by their type and parameters (for example mean and deviation for a Gaussian). At runtime, these are sampled, keyed from a unique name.
- To ensure the names remain unique between code branches (e.g, for each window independently), we compartmentalize the name spaces into *nodes*. These are stored in a tree parallel to the code branches.
- The sampled parameters are stored in a tree of nodes. This is serialized to disk (as a JSON file) named for the random seed of the root node.
- With some probability distribution, a certain node may differ from a sample to its parent. This mechanism supports shared parameters across the model.
- If the same model is rerun, we load the node-tree. As the model is executed we look up values in the tree: found parameters are returned (this provides reproducibility) otherwise they are sampled. We may have to perform mixed lookup and sampling if the code has changed, causing different values to be sampled (this provides robustness).
- As each node is created, a pseudo-random generator is created and initialized by a parameter sampled from the parent node's generator. This compartmentalized random increases robustness, as subsequent samples from the parent generator will not impact the children. The seed for the root node describes the whole scene and is computed as a hash of the current time and compute node.
- We track changes to our distributions through development using version control software and a continuous integration system. This allows us to track segmentation task accuracy as source code changes are made.

During the development of WinSyn, we iterated the parameter distributions based on assessed mIoUs on real training data, synthetic data, as well as label integrals (Figure 13). Careful examination of these results often reveals features that are under-performing. We found that evaluating labeling accuracy on synthetic hold-outs was useful for validating synthetic labeling and identifying bugs.

Figure 12. A sample of 27 laser-scanned clutter meshes from our collection of 278 (of which 38 are suitable for placement on the wall).

Rendering. We use the Cycles renderer [1], with 256 samples per pixel and the default Open Image Denoiser [11] at 512×512 pixel resolution to create our color images. The average time to generate a synthetic datum (color image and label map) was 49.8 seconds. The dominant aspect of the rendering was the geometry generation (mean 38.1 seconds), followed by the rendering of the color image (9.2 seconds), and finally the rendering of the labels (2.4 seconds). As in Figure 14 there was considerable variation in these values. We observe the time to generate an image was dominated by the geometry generation. The hardware used for these timings was an NVIDIA Tesla P100 with a shared Intel(R) Xeon(R) CPUs E5-2699 v3 @ 2.30GHz. This was a single GPU from a multi-GPU machine, so timings may have been affected by other users' workloads. However, it allowed us to distribute work in parallel over 12 nodes, and generate the entire dataset of 21,290 images within a day.

System. This synthetic image generation system is implemented in Python 3.10 within the Blender [5] 3.3 modeling package. We make use of a number of Blender's features including:

- Screen-space subdivision allows our procedural textures (e.g., brick) to generate geometry as a separate depth channel. This allows the textures to self-shadow (e.g., one brick casting onto a brick below), but can be GPU memory intensive.
- Geometry-nodes are utilized to create features that is contain repeated components and benefit from shared geometry instances. We use them to create roof tiles, dirt (leaves and litter) on the floor, blinds, and slats. Using the node-based 'language' to describe these is somewhat limiting, but allows faster development and lower memory use.
- Publicly available shaders and geometry nodes-trees. There are a wide number of marketplaces with content available for Blender for free or relatively low cost. How-

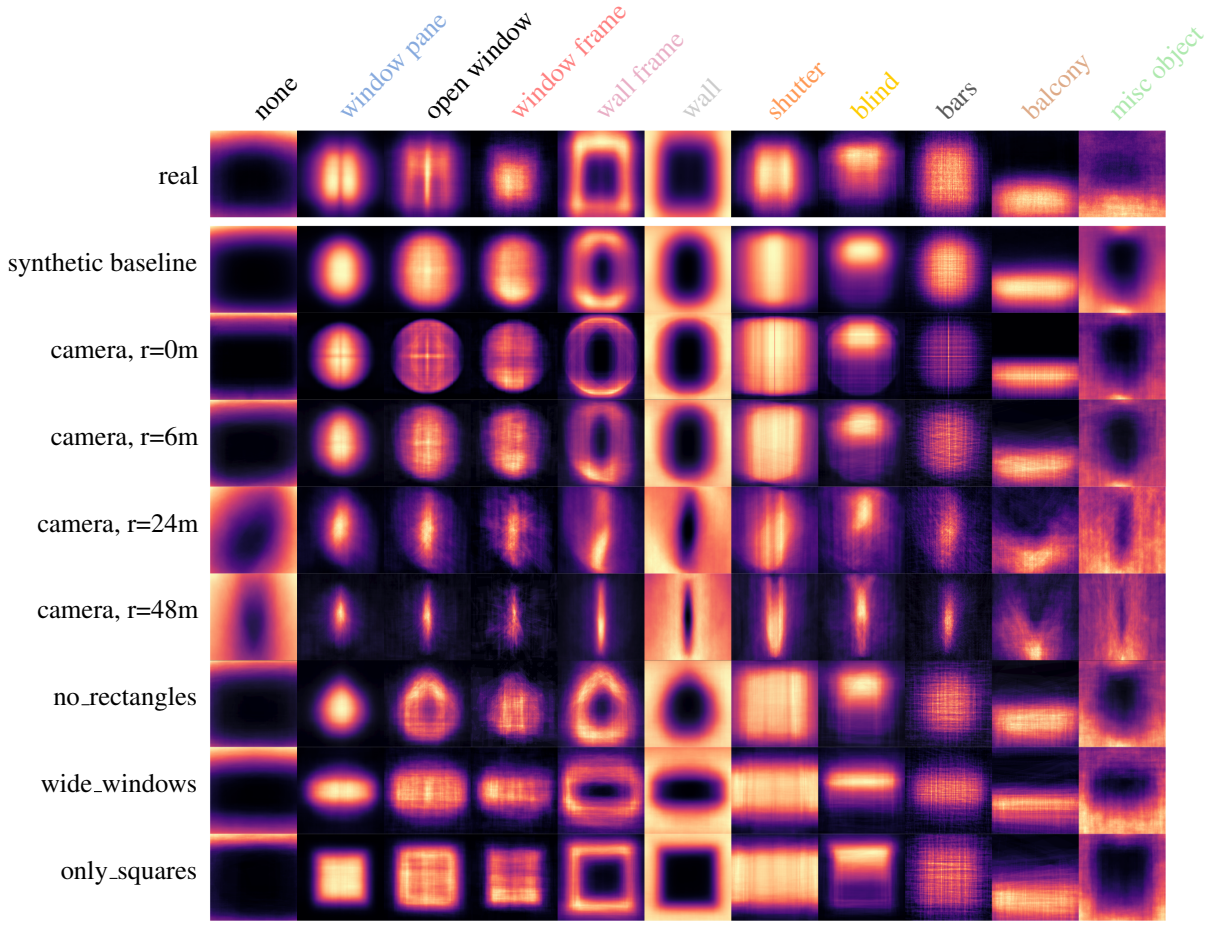


Figure 13. Label integrals for different datasets. For each label (columns) in each dataset (rows) we sum the label masks and normalize per-image. The top row is our ground truth labeled photographs ($n = 9,002$). The second row shows our baseline procedural model ($n = 21,290$). The following rows show various variations ($n = 2,048$) - the *camera location* variations (0m..48m), and *window geometry* experiments (no_rectangles, wide_windows, only_squares).

ever, the licenses may limit the downstream applications and distribution of the model.

7. Variations

One advantage of WinSyn is that the compact domain allows rapid experiments and iteration. Exploiting this, we create 64 synthetic datasets, each of size $n = 2,048$ and resolution 512×512 pixel, as variations of our baseline synthetic data with different sizes and mixes of real or synthetic data, geometry, textures, lighting, labels, and camera positions. These are used to train a segmentation network and test on our standard real data partition of $4,906$ photos. For comparison, the accuracy of the baseline synthetic model was on this partition was 32.58. We provide an analysis of these results as a demonstration of the testbed and to provide guidance to others creating synthetic procedural models.

7.1. Rendering samples

We evaluated the impact of samples per pixel (*spp*) on render quality. This was implemented by changing the parameters in the Cycles renderer, and disabling the neural denoiser that is now industry standard. All images (in this, an other variations) were stored uncompressed to avoid introducing additional artifacts into already noisy images. See the main paper for results.

7.2. Wall Materials

Here we increase the number of wall materials from 1 to 128, $w \in \{1...128\}$. All other objects use the baseline material distributions; the wall is assigned materials from

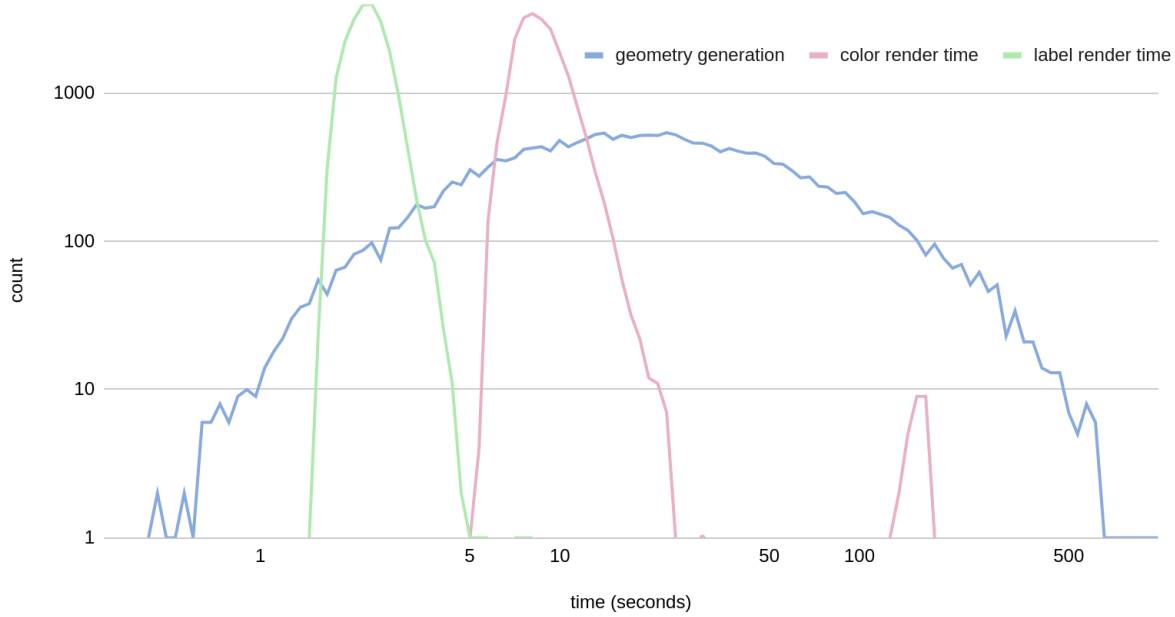


Figure 14. Generation time histogram for the synthetic dataset. Note the log-scale axes as the process was dominated by the geometry generation. Mean total time per sample was 49.8 seconds. The physics simulator ran within the geometry generator for timing purposes.

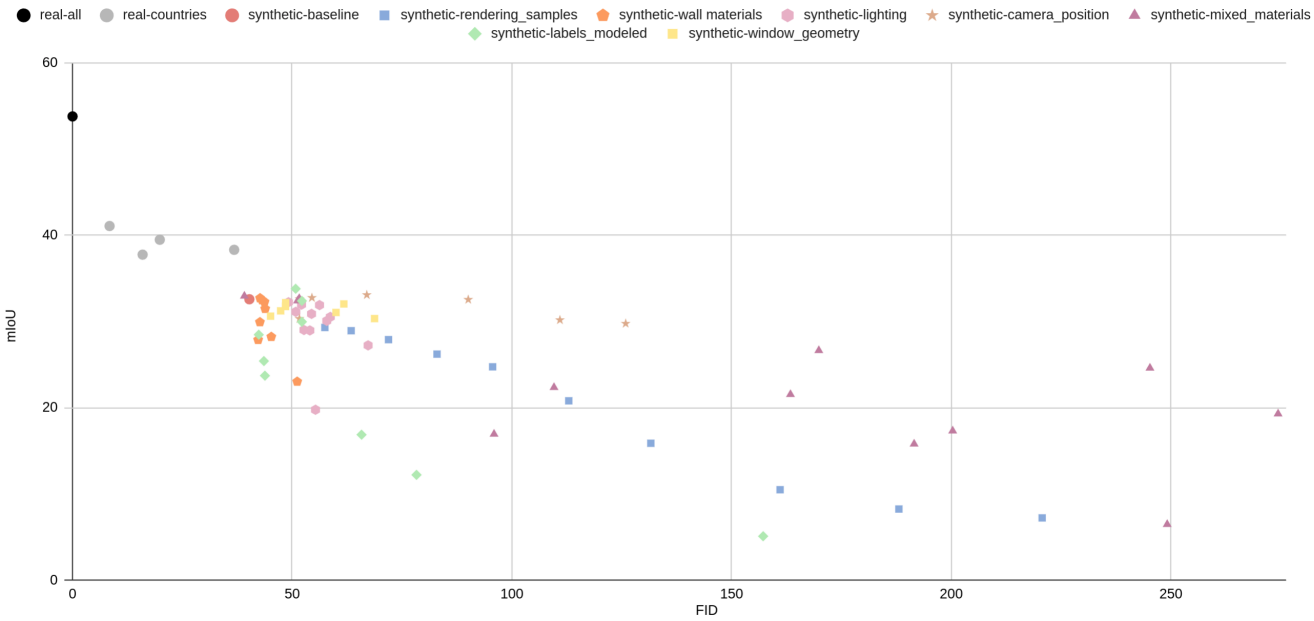


Figure 15. Relative mIoU and FID performance of each synthetic variation in comparison to real photographs. Each sequence of experiments has a different color. mIoU is measured against our standard real test-split. FID is measured against all 89,318 photo crops; in the case of countries, those countries themselves are excluded. For mIoU, train size is 2,048 for synthetic variations, 1,024 for individual countries, and 4,096 for all countries; the test split is our standard $4,906$. Exact values for the variations are provided in Figure 23.

a material library [6]. The results are shown in Figure 17, as well as the main paper; we note that although the single-

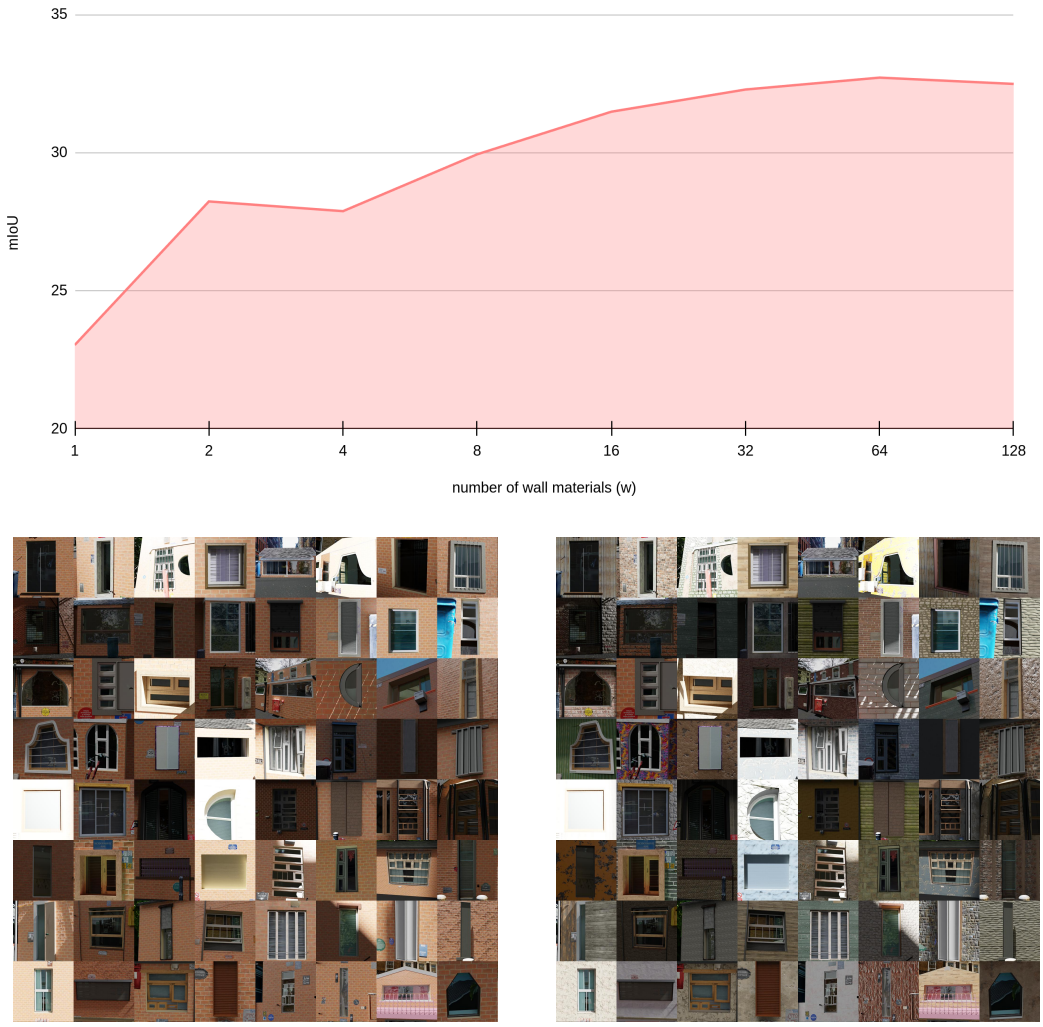


Figure 16. Top: segmentation task accuracy over datasets with different numbers of wall materials. Bottom: samples from the $w = 1$ (left) and $w = 128$ (right) variations.

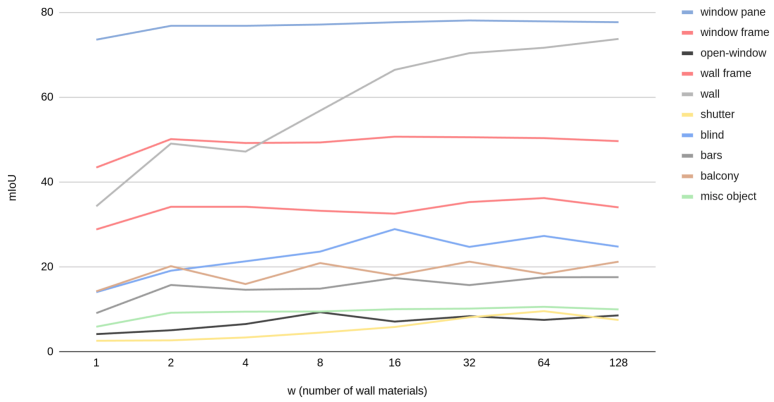


Figure 17. The impact of different numbers of discrete wall materials w.

class mIoU of wall is still improving at $w = 128$, the overall mIoU saturates due to the impact on other classes.

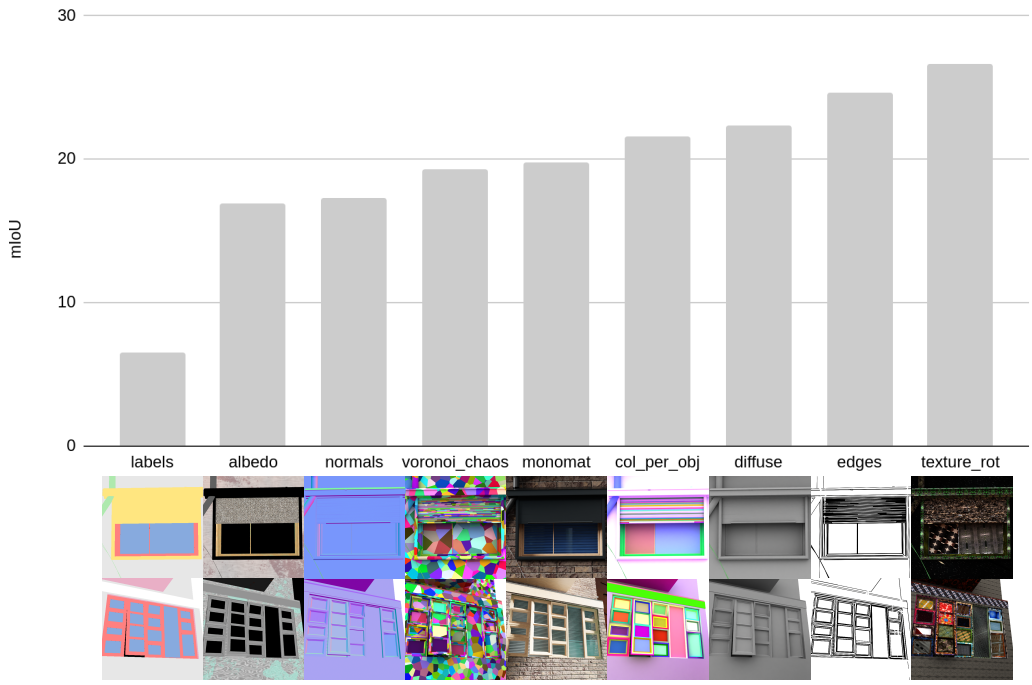


Figure 18. The impact of a variety of different materials on labeling task performance.

7.3. Mixed Materials

While the above materials results give a deep view of the impact of single-class materials, we also study a wide scenario - applying a wide variety of materials to all object classes. Four of these material variations use no lighting model (labels, albedo, normals, and lines). The remainder use a simple diffuse lighting model, with no direct sun (as Figure 18). The geometry is identical for each render. The material variations studied are:

- *labels*: The labels rendered from the synthetic dataset. These were included as a baseline for a 'bad' variation with label cohesion, but very low realism.
- *albedo*: The albedo pass from the renderer.
- *normals*: The screen-space normal map.
- *vornoi_chaos*: Each object has the same 3D vornoi-cell texture, but with different scale and offset parameters. Each cell has a random color.
- *monomat*: For each object type in the scene, we apply the same material across the whole dataset. The parameters for the procedural materials are fixed.
- *col_per_object*: Each object in the scene is a random color.
- *diffuse*: The entire scene has a single mat material with gray color.
- *edges*: An edge renderer is used to create a sketch-like image of the scene.
- *texture_rot*: For each object, we select one from a collection of 50 geometric textures, and apply random lighting and scale.

7.4. Lighting Models

The results from the lighting variations are shown in Figure 19. Each variation explores a different aspect of the lighting model. The variations modeled are:

- *albedo*: (as the Material variation) The albedo pass from the renderer. No other lighting terms.
- *phong_diffuse*: The diffuse term from the Phong lighting model with a gray material. No other lighting terms.
- *diffuse*: (as the Material variation) The entire scene has a single matt material with gray color.
- *night_only*: Only the night (less light from the sun, brighter internal lighting) mode is used. Dark images.
- *no_sun*: No directional light source in the scene.
- *no_bounce*: The path tracer terminates the trace after the first bounce.
- *fixed_sun*: The sun is always in the same location and size.
- *day_only*: Only use the day lighting model (no night).

The usual whitening across a dataset is performed on all variations before training and testing; but this post-process reduces useful color depth and is dataset-wide. For these lighting experiments, we also examine a per-image *exposure* pass within the render pipeline to preserve color depth and perform brightness equalization. This is similar to histogram equalization in that it adjusts the image brightness, but based on the central areas of the image, simulating the

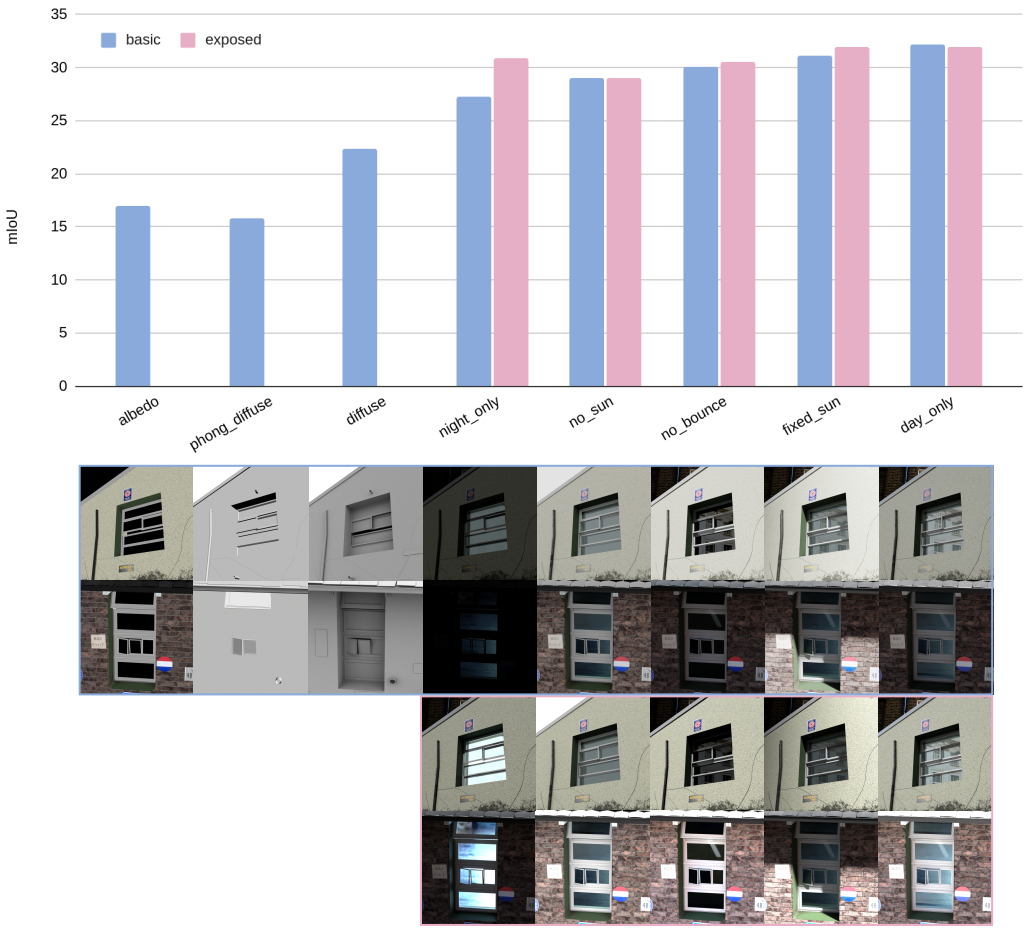


Figure 19. The impact of different lighting models on the labeling task mIoU.

auto-exposure mechanism in a digital camera. We can see that for poorly exposed dataset (such as *night_only*) this significantly improves the performance. However, for better balanced datasets (*day_only*) there is a slight performance degradation (Figure 19).

Generally, it is important to have realistically exposed images. The synthetic model has a night-time mode under the assumption that windows would look very different when lit from within; however, the night-time lighting setup was counter-productive, reducing task effectiveness. In contrast, a sun with multiple positions creating shadows was somewhat useful to the task.

7.5. Camera Position

Windows are a useful domain for studying camera position distribution as they have an obvious unambiguous canonical orientation. In this variation, we experimented with the distribution of camera positions. We used a simple model which sampled a camera position over a circle, of radius r meters, truncated at the floor plane (Figure 20, right). The circle is positioned 5 meters from the wall, directly in front of the window. As r increases, the majority of the circle area moves away from the window, and so the camera angle to the wall becomes very shallow in many samples. Larger circles have a large fraction of their area higher up, so create unusual camera angles not present in the photographic dataset. However, we observe that high r does not impact the segmentation task accuracy as strongly as we might expect (Figure 20, left). Note that extreme camera angles may be able to see windows perpendicular to the primary window on which the camera camera is targeted. We conclude that our model is only somewhat sensitive to the distribution of camera positions.

The baseline models removes clutter from the scene which occludes the camera view of the primary windows. These variations do not perform this occlusion check. At more extreme camera positions, there is a greater chance that objects (wall clutter, bay windows, or balconies) block

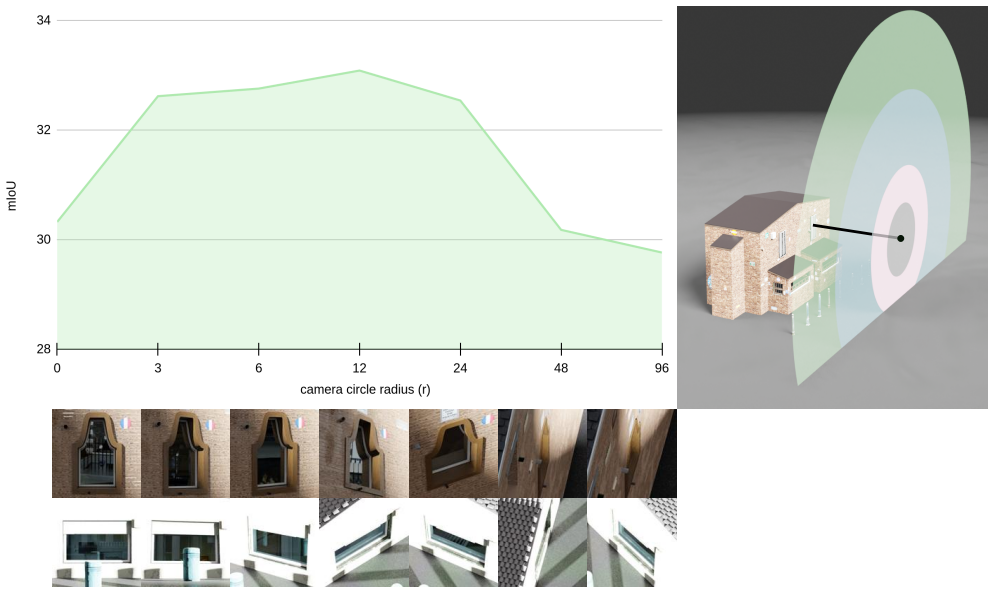


Figure 20. Left: the mIoU and average accuracies for different camera-sampling-circle radii, r. Right, top: The position of the circles' centers is $5m$ (black line) front of the top right window. For scale, the green circle has $r = 12m$. Bottom: samples from the different distances.

the view of the primary window.

7.6. Window Geometry

Here we study the impact of changing the distribution of parameters which impact the window geometry, as Fig 21. Note we do not have a uniform base geometry in this experiment as the distribution changes impacted scene-wide geometry and material choices. The label integrals for some of our parameter distributions are shown in Figure 13. The window geometry distributions variations are:

- *no_rectangles*: Only non-rectangular windows (circular, arched, and angled)
- *only_squares*: Only square windows.
- *no_splits*: No window-pane subdivision took place. Each window has a single pane.
- *only_rectangles*: Only rectangular windows.
- *single_window*: Only a single window created.
- *wide_windows*: The width/height parameters are adjusted to create wider, shorter windows.
- *mono_profile*: We reduce the number of all the extruded profiles in the system. These are used on window frame, sill, lintels, roof gutters. The set of profiles used for windows is reduced from the baseline (Fig 21, right) to a single profile.

7.7. Labels modeled.

Real-world diversity of even simple man-made objects such as windows is huge (as Figure 3). When developing a procedural model as a synthetic data generator, we must allocate software development effort effectively. Our model was developed approximately in order of label sizes: largest first. In this variation, we take the completed model, and perform ablations by removing each class in turn to study the impact of each class in a systematic manner. The main paper introduces the all-class mIoU and Figure 22 illustrates the per-class breakdown. We observe the diminishing returns in mIoU for the later labels, as the features get smaller and the larger choice of labels makes labeling harder.

The label-level ('lvl') variations are:

- *lvl1*: Only the wall geometry is present.
- *lvl2*: Add a window panes. It is not recessed.
- *lvl3*: Add a wall-frame and recess the window pane.
- *lvl4*: Add a window-frame and any splits to the window panes.
- *lvl5*: Add window shutters.
- *lvl6*: The balconies are added. These use the *balcony* for their guard rails and *wall-frame* labels on their lower parts.
- *lvl7*: Wall and floor clutter is added to the scene.
- *lvl8*: Add window blinds.
- *lvl9*: The final label to add we add is bars.

The full model includes additional features beyond lvl9, such as interior dressing, open windows, and windows-without-glass.

7.8. Summary

Figure 15 provides a summary of the variations' segmentation performances as well as the Frechét Inception Distance (FID) [8] to the ground truth. FID provides an estimate as

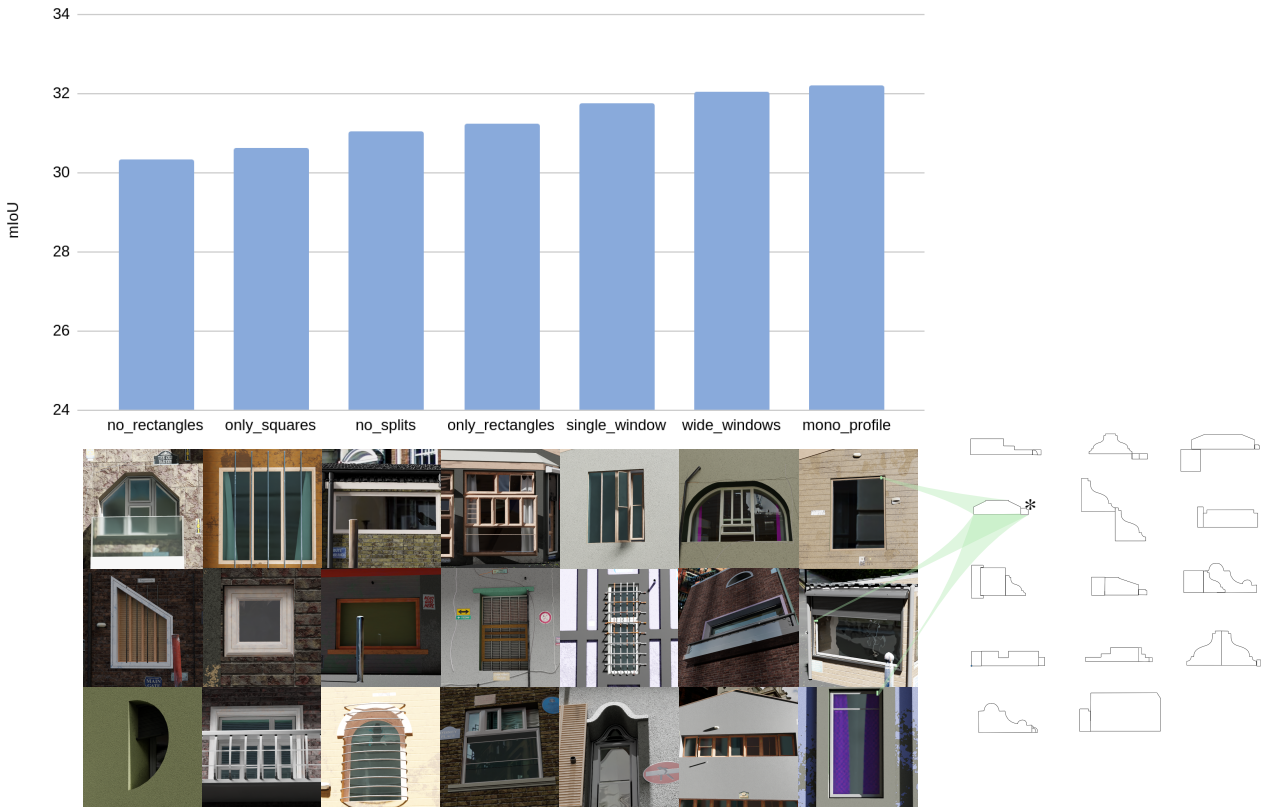


Figure 21. Top: The impact of changing the window shape and geometry on the mIoUs of the labeling task. Bottom left: examples of the $n = 2,048$ training examples in each variation. Bottom right: the full set of window-frame profiles for the baseline model; an asterisk marks the single profile used in the *mono_profile* variation. Note vertical axis starts at non-zero.

to the quality of the variation with respect to our real-world photos independent of segmentation task. We observe that different variations have widely different increases in segmentation task performance for FID improvements. That is, FID improvements are an inconsistent indicator of task performance between different variations, but somewhat useful within a single variation. We see this in Figure 15 as different sequences of variations approach the baseline at different angles – some variations allow large increases in performance while minimally impacting FID (such as *labels modeled*), while others have a large impact in FID, but much lower improvements in mIoU (*camera position*).

At the end of this Section, in Figure 23, we show samples from the images created for each variant dataset explored in the paper and provide their mIoU and FID against the ground truth. We continue to describe how each variation is modified from the baseline described above.

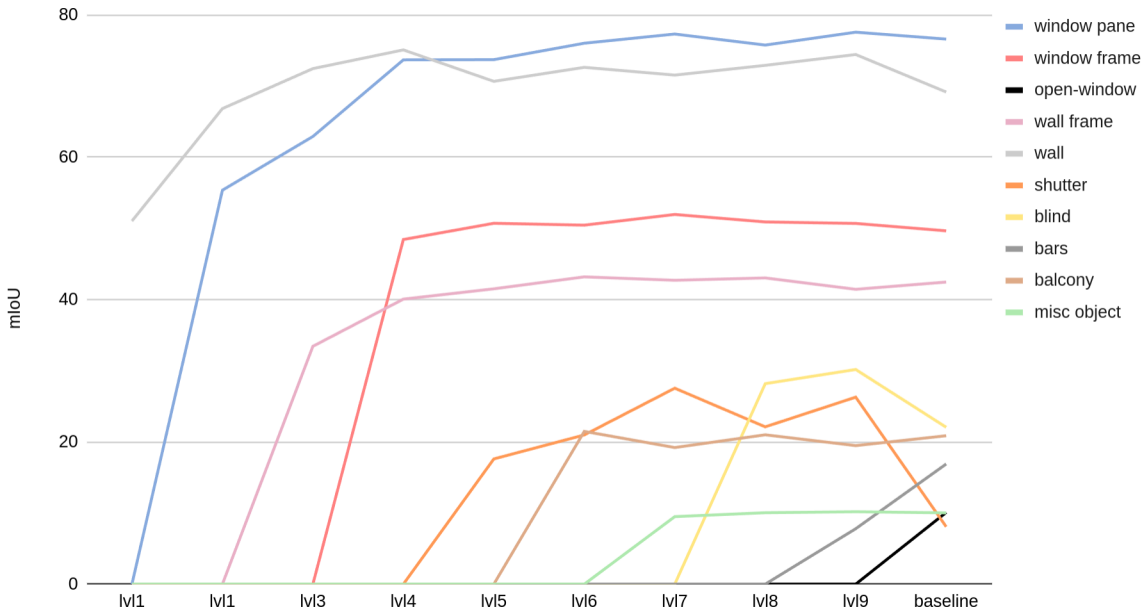


Figure 22. The impact on the per-class IoU of modeling each label. We note that sometimes modeling more labels will harm segmentation quality.

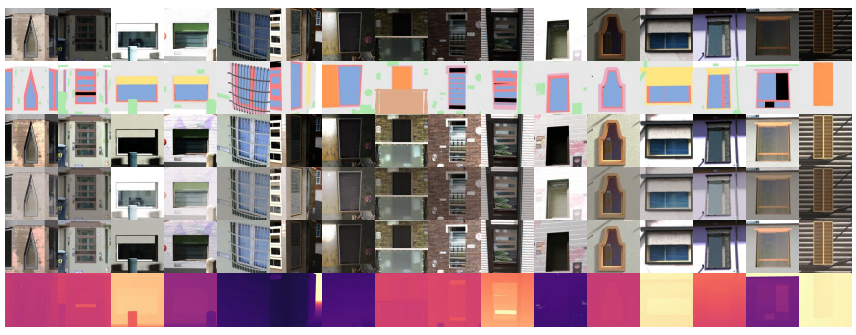
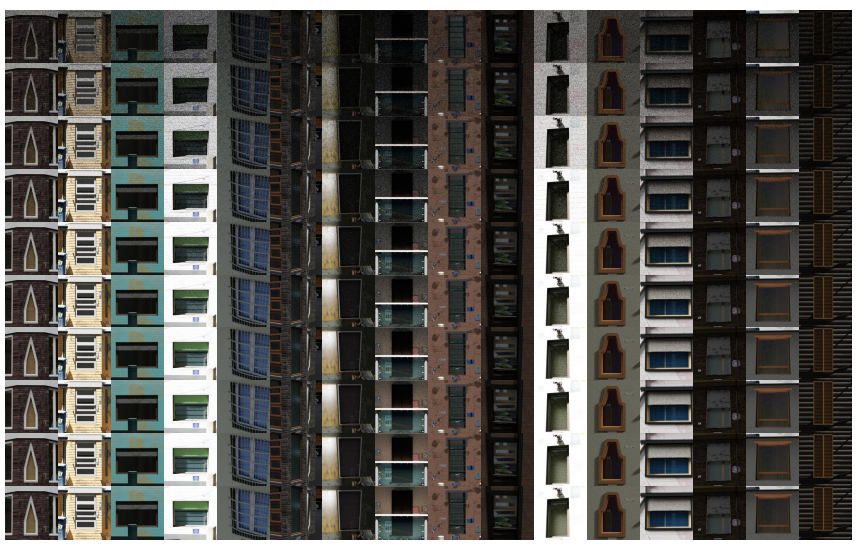
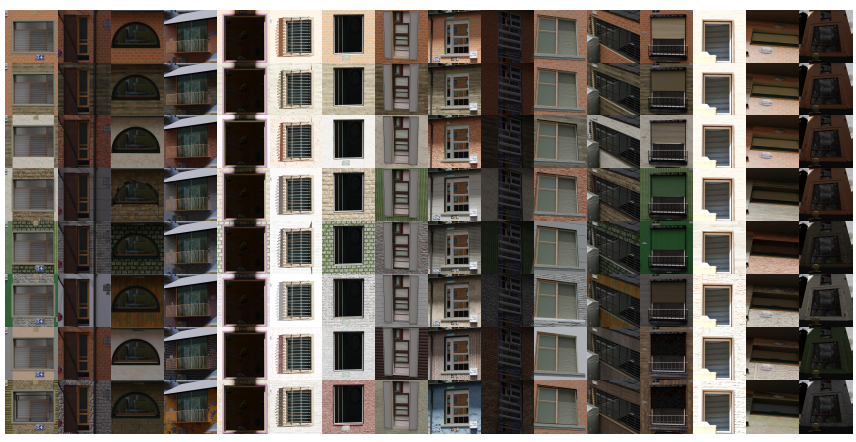
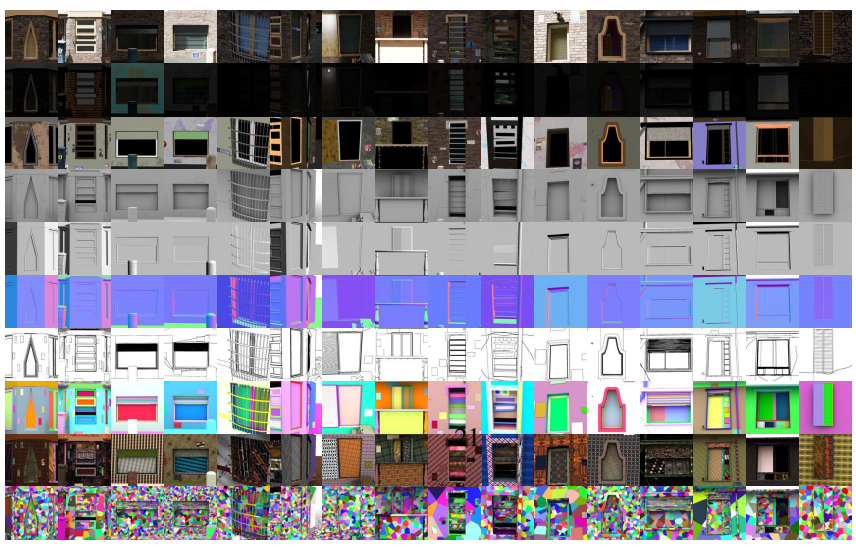
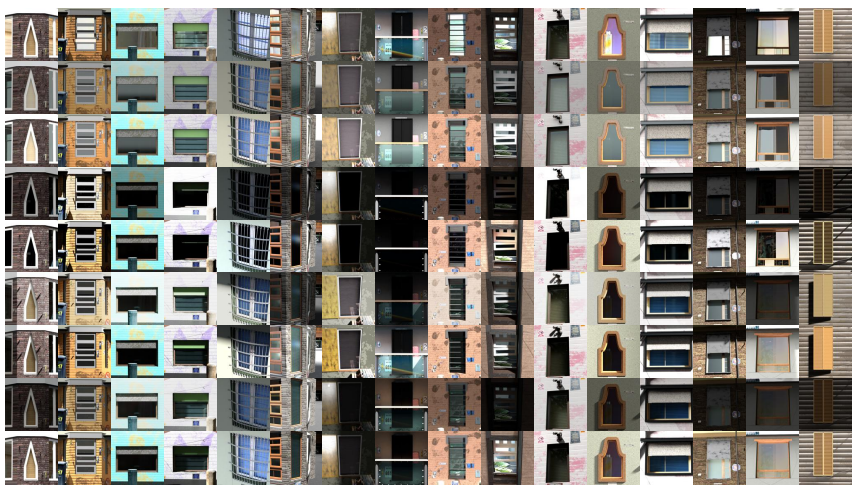
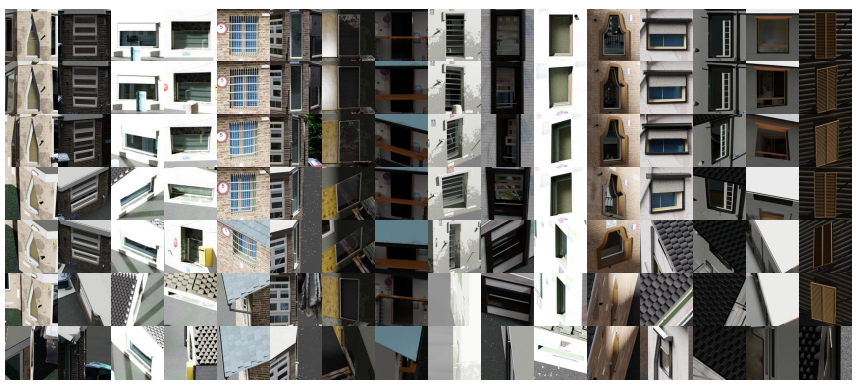
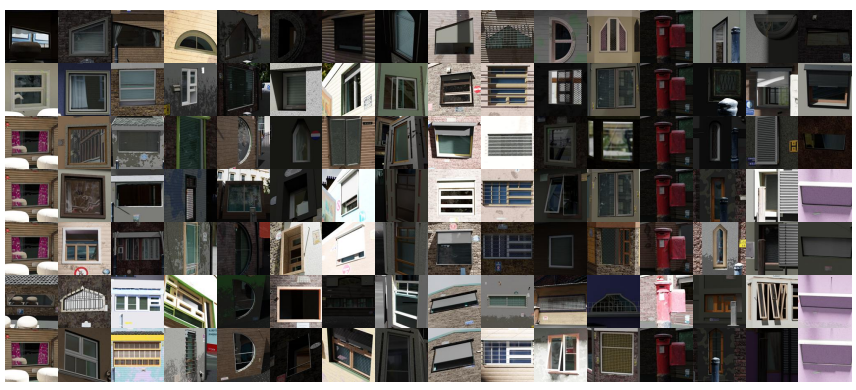
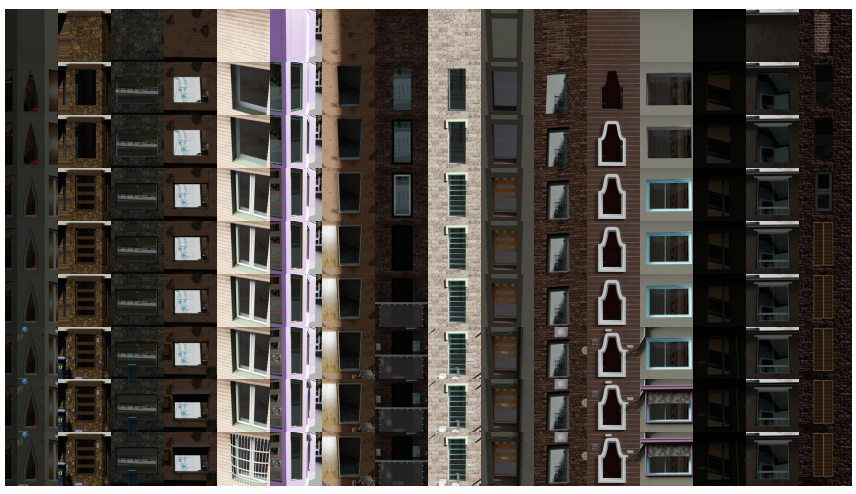
name	random samples	mIoU	FID
baseline		32.59	40.24
labels		6.48	249.15
exposed(baseline)		32.41	51.21
histo(baseline)		32.96	39.11
histo(exposed(baseline))		32.68	51.60
baseline_depth		na	na
1spp		7.23	220.69
2spp		8.26	188.08
4spp		10.50	161.04
8spp		15.89	131.62
16spp		20.81	112.94
32spp		24.75	95.62
64spp		26.22	82.96
128spp		27.91	71.92
256spp		28.94	63.41
512spp		29.31	57.43
wall materials,w=1		23.04	51.13
wall materials,w=2		28.23	45.24
wall materials,w=4		27.88	42.25
wall materials,w=8		29.94	42.65
wall materials,w=16		31.48	43.84
wall materials,w=32		32.28	43.69
wall materials,w=64		32.72	42.68
wall materials,w=128		32.49	43.15
monomat		19.77	55.30
night_only		27.24	67.27
baseline_albedo		16.95	95.94
diffuse		22.36	109.58
phong_diffuse		15.81	191.54
normals		17.32	200.32
edges		24.61	245.22
col_per_obj		21.55	163.40
texture_rot		26.65	169.85
voronoi_chaos		19.30	274.39
exposed(night_only)		30.89	54.39
no_sun		29.02	52.67
exposed(no_sun)		28.97	54.01
no_bounce		30.08	57.87
exposed(no_bounce)		30.54	58.67
fixed_sun		31.13	50.82
exposed(fixed_sun)		31.91	56.21
day_only		32.24	49.14
exposed(day_only)		31.96	52.11
camera, r=0m		30.33	51.64
camera, r=3m		32.62	51.68
camera, r=6m		32.76	54.49
camera, r=12m		33.09	66.97
camera, r=24m		32.55	90.06
camera, r=48m		30.18	110.93
camera, r=96m		29.77	125.89
no_rectangles		30.34	68.74
only_squares		30.63	45.07
no_splits		31.06	59.96
only_rectangles		31.24	47.37
single_window		31.74	48.47
wide_windows		32.04	61.75
mono_profile		32.20	48.51
lvl1		5.10	157.19
lvl2		12.22	78.29
lvl3		16.88	65.77
lvl4		23.73	43.79
lvl5		25.42	43.56
lvl6		28.48	42.39
lvl7		29.98	52.20
lvl8		32.40	52.20
lvl9		33.80	50.78

Figure 23. Each row shows samples from the 2,048 images in each variant, with the associated the mIoU and FID. The mIoU is calculated against the test partition of the labeled data, while the FID is calculated against all cropped photos; these are not shown for the depth 'variation'. We denote the renderer's exposure pass as *exposed()* and histogram equalization as *histo()*

.

References

[1] *Cycles: Open Source Production Rendering*. Blender Foundation, 2023. 12

[2] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *Acm Siggraph*, pages 1–7. vol. 2012, 2012. 8

[3] Prem Chedella and Kelly Tom. Street sign dataset, 2022. 9

[4] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018. 2, 8, 9

[5] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2023. 12

[6] Petr Dlouhý. *BlenderKit - Improved Blender workflow with new ideas.*, 2023. 14

[7] Rafael C Gonzales and Paul Wintz. *Digital image processing*. Addison-Wesley Longman Publishing Co., Inc., 1987. 3, 8

[8] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 18

[9] Lukas Hoyer, Dengxin Dai, and Luc Van Gool. Daformer: Improving network architectures and training strategies for domain-adaptive semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9924–9935, 2022. 3

[10] Lukas Hoyer, Dengxin Dai, Haoran Wang, and Luc Van Gool. Mic: Masked image consistency for context-enhanced domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11721–11732, 2023. 3, 8

[11] Intel. *Intel® Open Image Denoise*. Intel, 2023. 12

[12] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4551–4560, 2019. 11

[13] Pascal Mueller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM Trans. Gr.*, 25(3):614–623, 2006. 3, 12

[14] Zhiliang Peng, Li Dong, Hangbo Bao, Qixiang Ye, and Furu Wei. Beit v2: Masked image modeling with vector-quantized visual tokenizers, 2022. 2, 8

[15] Erroll Wood, Tadas Baltrusaitis, Charlie Hewitt, Sebastian Dziadzio, Thomas J. Cashman, and Jamie Shotton. Fake it till you make it: face analysis in the wild using synthetic data alone. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3661–3671. IEEE, 2021. 3, 8, 10