Supplemental Materials for Activity-centric Scene Synthesis for Functional 3D Scene Modeling

Matthew Fisher Manolis Savva Yangyan Li Pat Hanrahan Matthias Nießner

Stanford University

1 Introduction

This document provides additional details about the implementation of the scene synthesis algorithm of Fisher et al. [2015], the associated activity detection and plane extraction methods, as well as the types of data provided in our scene, model, and scan databases. Section 2 lists and discusses the parameters used in our synthesis pipeline. The methodology and parameters used in activity detection and plane extraction are discussed in Sections 3 and 4. The details of our database are given in Section 5.

2 Synthesis Parameters

2.1 Scene Template Parameters

- ColumnSize (Section 4.1 of the paper) — This parameter sets the X and Y spatial extent of each vertical column in the coarse geometric representation. All results use a value of $5\,\mathrm{cm}$, as this is the approximate accuracy of our 3D scans. A smaller value is appropriate with scans that contain fine-scale detail obtained from a more accurate sensor. Larger values can cause synthesized scenes to have poor alignment with objects such as monitors or chairs as the desired details are smoothed over by the coarse discretization.
- σ_s, σ_r (Section 6 of the paper) — These parameters control how strongly the geometry of the synthesized scene is penalized for deviating from either the supporting plane or the residual geometry height observed in the scan. "Supporting plane height" refers to the height of the top-most supporting plane (if any), while "residual height" refers to the height of geometry on top of the supporting plane. We use a value of $15\,\mathrm{cm}$ for both parameters, thus controlling how closely retrieved objects must match the scanned geometry. Small values for these parameters are likely to reduce the model diversity of synthesized scenes because otherwise valid models will be rejected if their dimensions do not match the scan closely enough. Conversely, large values for these parameters are likely to increase model diversity but also deviate more strongly from the geometry of objects in the input scan.

2.2 Agent Parameters

- AgentSittingHeight — This parameter specifies the distance along the z-axis between the agent's eyes and its hip-support plane assuming the agent is sitting. We use a value of $0.9\,\mathrm{m}$, corresponding to an average adult.
- AgentStandingHeight — This parameter specifies the distance along the z-axis between the agent's eyes and the floor plane when the agent is standing. We use a value of $1.55\,\mathrm{m}$, corresponding to an average adult.
- AgentShoulderDisplacement — This parameter specifies the distance between the agent's shoulders and the agent's head. We use an offset of $0.25\,\mathrm{m}$ below the eyes along the z-axis and an offset by $0.25\,\mathrm{m}$ to the left or right of the agent.

2.3 Sampling Parameters

The parameters detailed in this section are introduced in the pseudocode of Section 8.2.

- k_m — The number of models sampled from the backing model database at each iteration of the generation algorithm. Using a value of k_m that exceeds the numbers of models of that category from the database will cause the algorithm to consider placing all models in the scene. This will typically result in only the single, best scoring model being inserted which can reduce the model diversity in the synthesized scenes. Conversely, choosing a small value of k_m may overly restrict the number of models considered causing the chosen model to be a poor geometric or interaction fit to the scene. All results in the paper use $k_m = 5$.
- k_l — The number of locations and rotations sampled as candidate placement locations for each object. We choose k_l to sample one point with a random rotation every $1\,\mathrm{cm}^2$, to a maximum of 10000 points per candidate model. Generally, k_l should be chosen as large as possible although larger values will linearly increase the time it takes to generate a scene.

3 Activity Detection

In order to detect activities in the given input scenes, we apply an approach based on prior work called SceneGrok [Savva et al. 2014]. We use the annotated scene and recording dataset released by the authors to train a set of SceneGrok action map classifiers[1]. All parameters are set as reported in the paper; most importantly, 100 dictionary centroids are used for the segment dictionary activation function.

After the trained classifiers are obtained, we create an action map for each given input scene and action label a. We sample 5000 random x-y positions within the bounding box of the input scene and for each we evaluate the likelihood of the action by positioning a human pose (labeled with the action a in the training set) at 36 orientations in 10 degree increments, parameterized by an angle θ. Following the original approach, the poses are sampled at random from the set of poses with the given label. Finally, we store within each sample point a tuple (x, y, θ, p_a) where the first three

[1] `http://graphics.stanford.edu/projects/scenegrok/`

elements describe the position and orientation of the sampled pose, and the final element is the evaluated likelihood p_a of the action a occurring given the pose at that position and orientation. As this is a classification score between 0 and 1, with 0.5 indicating classifier uncertainty, we threshold all values less than 0.5 to 0, preventing proxy agents from being sampled in these regions.

4 Plane Extraction

The method for dominant plane extraction from 3D point clouds is adopted from Mattausch et al. [2014]; the authors' implementation is available on their website[2]. This is a clustering method based on region growing, with careful design for speedup and robustness. Two important keys in region growing clustering are where to start and when to stop. The method starts with picking growing seeds by an uniform grid sampling, with a grid size of 5 cm. Since we obtain point clouds from truncated signed distance functions, the normal of each point is reliable, and we can compute robust principal curvatures from the point coordinates and normal (we use a search radius of 5 cm). Growing seeds with large curvatures are pruned from being seeds, as points in curvy areas are unlikely to be good starting points for extracting large planes. Each seed has its position and normal, from which the plane parameter can be determined. Then, we progressively traverse the points and add them into the plane if (a) the angle between the plane normal and point normal is less than 20 degrees, and (b) the distance between the point and the plane is less than 5 cm. Once the number of points belong to a growing plane doubles, the plane parameters are re-estimated by least square fitting to the included points. This re-estimation strategy stabilizes the growing process, and outputs better results. Finally, we compute the area of the collected planes, and discard planes with areas less than 0.04 m^2. Note that our pipeline only cares about dominant planes, thus we can set rather loose parameters to safely discard suspicious planes.

5 Database Details

Here, we summarize the different types of data used by our system and how they are acquired. Note that all of this information can be found on our project page[3]; code for accessing and reading the database files can be also found online[4]

5.1 Real-world Data

In order to obtain a coarse 3D reconstruction, we use Kinect v1.0 sensor to scan a target, static environment. The resulting depth and color frames are fused together into a single coordinate system using Voxel Hashing [Nießner et al. 2013], and marching cubes is used to extract a polygonal mesh from the signed distance function[5]. The resulting mesh is transformed so that the floor is at $z = 0$, oriented upwards along the positive z-axis. We provide this oriented mesh for each scan as a Wavefront OBJ file.

Next, activity detection is performed as described in Section 3 of the supplemental materials. We provide the resulting activity maps for each activity in the coordinate frame of the OBJ file. We also provide a sampling of the localized, oriented, and labeled agents in each scene.

[2]`http://www.ifi.uzh.ch/vmml/publications/ObjDetandClas.html`

[3]`http://graphics.stanford.edu/projects/actsynth/`

[4]`https://github.com/techmatt/actsynth/`

[5]`https://github.com/nachtmar/VoxelHashing/`

5.2 Virtual Data

For the virtual scene models, we employ the Stanford 3D scene database which is publicly available[6]. We augment the scene database with additional scenes in order to reflect the new activities in our database. Note that we provide the augmented database of 3D scenes on our project page. Each scene is a collection of 3D objects, where each object has a reference to a model in the 3D model database, a coordinate frame of this model in the space of the 3D scene, and a parent static-support object. Each model may use a different uniform scale of the input 3D model. We provide the entire 3D model database that the scenes are drawn from as Wavefront OBJ files along with associated 2D textures as JPG files (these textures are only used for rendering and are not needed by the synthesis method).

We annotate each scene with discrete, oriented virtual agents each labeled with a specific activity. Each object in each virtual scene is bound to zero or more agents indicating that object is used as part of that agent's behavior. We provide the agent annotations and agent-object bindings for each virtual scene. Finally, we provide the annotations for each model used in the 3D scene database with discretely-sampled interaction maps. The format of these scene and interaction map annotations is specified on the project website.

References

FISHER, M., SAVVA, M., YANGYAN, L., HANRAHAN, P., AND NIESSNER, M. 2015. Activity-centric scene synthesis for functional 3d scene modeling. *ACM Transactions on Graphics (TOG).*

MATTAUSCH, O., PANOZZO, D., MURA, C., SORKINE-HORNUNG, O., AND PAJAROLA, R. 2014. Object detection and classification from large-scale cluttered indoor scans. *Computer Graphics Forum 33*, 2, 11–21.

NIESSNER, M., ZOLLHÖFER, M., IZADI, S., AND STAMMINGER, M. 2013. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG) 32*, 6, 169.

SAVVA, M., CHANG, A. X., HANRAHAN, P., FISHER, M., AND NIESSNER, M. 2014. Scenegrok: Inferring action maps in 3D environments. *ACM Transactions on Graphics (TOG) 33*, 6.

[6]`http://graphics.stanford.edu/projects/scenesynth/`