Convex Decomposition of Indoor Scenes

Vaibhav Vavilala  
UIUC

David Forsyth  
UIUC

Abstract

We describe a method to parse a complex, cluttered indoor scene into primitives which offer a parsimonious abstraction of scene structure. Our primitives are simple convexes. Our method uses a learned regression procedure to parse a scene into a fixed number of convexes from RGBD input, and can optionally accept segmentations to improve the decomposition. The result is then polished with a descent method which adjusts the convexes to produce a very good fit, and greedily removes superfluous primitives. Because the entire scene is parsed, we can evaluate using traditional depth, normal, and segmentation error metrics. Our evaluation procedure demonstrates that the error from our primitive representation is comparable to that of predicting depth from a single image.

1. Introduction

Primitive based abstractions of geometry offer simpler reasoning – it is easier, for example, to deal with a fridge as a cuboid than attend to the detailed curves around a corner. But obtaining primitive representations that abstract usefully and accurately has been hard. Parsing isolated objects into parts has seen considerable recent progress, but parsing indoor scenes into collections of primitives has remained difficult. We describe a method to decompose indoor scenes into convex primitives which largely describe the scene. Our method is trained on labeled RGBD + semantic segmentation maps, and the predicted decomposition can be refined on depth + segmentation maps that are predicted from a suitable pretrained network.

Our method draws on a long history of fitting primitives to objects with two main methods (review Sec. 2). A descent method chooses primitives for a given geometry by minimizing a cost function. These methods are plagued by local minima and robustness issues. A regression method uses a learned predictor to map geometry to primitives and their parameters. These methods can pool examples to avoid local minima, but may not get the best prediction for a given input. Our method uses regression to predict an initial decomposition, then applies descent to polish the primitives for each particular test geometry. We show that both steps are critical to achieving a good fit.

It can be hard to tell if a primitive based representation captures the abstraction one wants. But for a representation to be useful, it must have three necessary properties:

1. Accuracy: The primitives should represent the original geometry. As Section 4 demonstrates, we do not need to create metrics specifically for scene parsing and instead existing evaluation procedures from depth/segmentation prediction literature are sufficient.

2. Segmentation: Generally, the decomposition should “make sense.” For isolated objects, this means that each primitive should correspond to an object part (which is difficult to evaluate). For scenes, this means that each primitive should span no more than one object. An object might be explained by multiple primitives, but a primitive should not explain multiple ob-
3. **Parsimony**: In line with previous geometric decomposition work, we desire a representation with as few primitives as possible. In this work, we show that our method can vary the number of primitives per-scene by initially starting with a fixed number, and allowing a post-process to find and remove unnecessary convexes.

Our contributions are:

1. Our primitive decomposition method for indoor scenes is a simple procedure that works well on established and novel metrics on the benchmark NYUv2 dataset. In particular, we are the first, to our knowledge, to evaluate primitive decompositions using standard metrics for depth, normal, and segmentation.

2. We show our mixed strategy offers drastic advantages over either descent or regression methods in isolation. The key insight is that convex fitting to noisy indoor scenes is extremely difficult via pure optimization, but very good start points can be generated via regression.

3. We are the first to use segmentation labels to drive primitive generation.

2. Related Work

There is a huge early literature on decomposing scenes or objects into primitives for computer vision. Roberts worked with scenes and actual blocks [42]; Binford with generalized cylinder object descriptions [4]; and, in human vision, Biederman with geon descriptions of objects [3]. A representation in terms of primitives offers parsimonious abstraction (complex objects could be handled with simple primitives; an idea of broad scope [7]) and natural segmentation (each primitive is a part [3, 4, 49]). Desirable primitives should be easily imputed from image data [35, 43], and allow simplified geometric reasoning [37]. Generally, fitting was by choosing a set of primitives that minimizes some cost function (descent).

There has been much focus on individual objects (as opposed to scenes). Powerful neural methods offer the prospect of learning to predict the right set of primitives from data – a regression method. Such methods can avoid local minima by predicting solutions for test data that are “like” those that worked for training data. Tulsiani et al. demonstrate a learned procedure that can parse 3D shapes into cuboids, trained without ground truth segmentations [48]. Zou et al. demonstrate parses using a recurrent architecture [53]. Liu et al. produce detailed reconstructions of objects in indoor scenes, but do not attempt parsimonious abstraction [32]. There is alarming evidence that 3D reconstruction networks rely on object identity [47]. Deng et al. (CVXNet) recover representations of objects as a union of convexes from point cloud and image data, again without ground truth segmentations [8]. Individual convexes appear to map rather well to parts. An early variant of CVXNet can recover 3D representations of poses from single images, with reasonable parses into parts [9]. Meshes can be decomposed into near convex primitives, by a form of search [52]. Part decompositions have attractive editability [19]. Regression methods face some difficulty producing different numbers of primitives per scene (CVXNet uses a fixed number; [48] predicts the probability a primitive is present; one also might use Gumbel softmax [23]). Primitives that have been explored include: cuboids [5, 13, 33, 48, 41, 44, 46, 27]; superquadrics [2, 22, 36]; planes [7, 30]; and generalized cylinders [35, 54, 29]. There is a recent review in [12].

Another important case is parsing outdoor scenes. Hoiem et al. parse into vertical and horizontal surfaces [20, 21]; Gupta et al. demonstrate a parse into blocks [14]. Yet another is indoor scenes. From images of indoor scenes, one can recover: a room as a cuboid [16]; beds and some furniture as boxes [17]; free space [18]; and plane layouts...
Figure 3. An overview of our inference procedure. An RGBD image and segmentation map is the input to an encoder-decoder. A series of losses supervise the training process, requiring samples labeled as “inside” or “outside.” The network tries to adjust the convexes to classify the points correctly. The network output can be optionally refined by greedily removing primitives that do not increase the loss, splitting convexes (effectively trading off parsimony for granularity) and finally directly optimizing the convex parameters with respect to the losses. This refinement procedure accepts a segmentation map and depth map (either GT or inferred e.g. [40, 39, 6]). Note that the network is unaware of the refinement process during training - it just tries to predict the best decomposition possible via the losses. The final decomposition can be visualized by ray marching from the original viewpoint and labeling each convex with a different color. Unlike previous methods, we can evaluate our approach against the original depth map by using traditional depth and normal error metrics.

3. Method - Convex Decomposition

At runtime, we seek a collection of primitives that minimizes a set of losses 3.1. We will obtain this by (a) passing inputs into a network to predict a start point then (b) polishing these primitives. The network is trained to produce a fixed length set of primitives that has a small loss 3.4. Polishing uses a combination of descent and greedy strategies 3.2. We summarize our procedure in Fig. 3.

3.1. Losses

Our losses use samples from depth 3.3. We begin with the losses and hyperparameters of [8]. A signed distance function of each point $\Phi(x)$ against every halfplane is defined using logsumexp to facilitate smooth learning. An indicator function based on sigmoid outputs a continuous classification score per-point between $[0, 1]$ effectively indicating whether a sample is outside all convexes or inside at least one. The primary loss is a sample loss that encourages the convexes to classify each sample correctly. We found the remaining auxiliary losses in the paper to be beneficial and adopted them with default hyperparameters. Additionally, in our early experiments we noticed long and thin convexes generated occasionally during training, whereas the types of convexes we’re interested in tend to be cuboidal. Thus we modify the unique parametrization loss to penalize small offsets:

$$L_{\text{unique}} = \frac{1}{H} \sum_h ||d_h||^2 + \frac{1}{H} \sum_n ||1/d_n||^2$$

where $H = 6$ is the number of halfplanes per convex.
Figure 4. Results from our convex decomposition method (third row) as compared with prior work from [26] (second row), which does not model the whole input. Using our data generation strategy and hybrid method, our primitives obtain better coverage. Our refinement and pruning process started from a fixed number of convexes (24 in this example) and was able to represent these scenes with an average of 14 convexes.

Manhattan World Losses We introduce three auxiliary losses to help orient the convexes in a more structured way. We have the network predict 9 additional parameters $\mathcal{M} \in \mathbb{R}^{3 \times 3}$ encoding a Manhattan world, and encourage orthonormality with the loss:

$$L_{ortho} = \frac{1}{9} \sum (I - \mathcal{M}' \mathcal{M})^2$$

(2)

where $\mathcal{M}'$ is the transpose, and $I$ is the identity. The weight of this loss is 10.

Next, we must introduce an alignment loss that encourages the faces of each convex to follow the Manhattan world (which forces us to fix the number of planes per convex to 6 for simplicity). Let $\mathcal{N}$ be the $3 \times 6$ matrix containing the predicted normals of a given convex as its columns. Let $\mathcal{W} = [\mathcal{M}; -\mathcal{M}]$ be the $3 \times 6$ matrix capturing the Manhattan world basis and its negative. The second auxiliary loss is given by maximizing the element-wise dot product between $\mathcal{N}$ and $\mathcal{W}$:

$$L_{align} = 1 - \frac{1}{6} \sum \mathcal{N} : \mathcal{W}$$

(3)

We normalize the Manhattan world vectors in $\mathcal{M}$ before applying $L_{align}$, and set its weight to 1. This loss effectively enforces an order for the normals within each convex.

Finally, we establish an explicit ordering of the convexes themselves, encouraging the network to generate them in order of volume. Since the predicted convexes are approximately cuboids, the volume of a convex can be estimated with $(d_0 + d_3) \times (d_1 + d_4) \times (d_2 + d_5)$ where $d_i$ is the offset for halfplane $i$. Given a vector $\mathcal{V} \in \mathbb{R}^K$ of volumes where there are $K$ convexes at training time, we can encourage the network generate convexes in order of volume:

$$L_{vol} = \frac{1}{K} \sum ReLU(\mathcal{V}[1:] - \mathcal{V}[0:-1])$$

(4)

The weight of this loss is 1. In practice, we observed that the first (and largest convex) was almost always the floor. Downstream applications may benefit from having the convexes presented sorted by size. As we show in our supplement, the volume loss has an approx. neutral effect on the error metrics but did improve parsimony.

Segmentation Loss We construct a loss to encourage the convexes to respect segmentation boundaries. Ideally, each face of each convex (or, alternatively, each convex) spans one label category in the image. To compute this loss, we obtain a smoothed distribution of labels spanned by a face (resp. convex), then penalize the entropy of the distribution. This loss penalizes convexes that span many distinct labels without attending to label semantics. We use the standard 40 NYUv2 labels, with one background label.

To compute the loss for convexes, write $\mathcal{C}$ for the $K \times N$ matrix representing the indicator function response for each inside surface sample for each convex, and $\mathcal{L}$ for the $N \times 41$ one-hot encoding of the segment label for each sample. Then each row of $\mathcal{C} \mathcal{L}$ represents the count of labels within each convex, weighted so that a sample that is shared be-
between convexes shares its label. We renormalize each row to a distribution, and compute the sum of entropies. To compute the loss for convex faces we obtain \( C_f \), \((6K) \times N\) matrix, where each row of \( C_f \) corresponds to a face. For the row corresponding to the \( i \)'th face of the \( k \)'th convex, we use the \( k \)'th row of \( C \) masked to zero if the closest face to the sample is not \( i \). Then each row of \( C_f \| L \) represents the count of labels spanned by each face, weighted so that a sample that is shared between faces shares its label. We renormalize each row to a distribution, and compute the sum of entropies.

\[
L_{\text{entropy}} = \frac{1}{6K} \sum \text{entropy}(C_f L)
\]  

We train with GT segmentation labels from NYUv2. During refinement, we can either use a GT segmentation map if available, or an inferred map (we use [6]). We experimentally weight \( L_{\text{entropy}} \) to 1.

### 3.2. Polishing

Our network produces reasonable decompositions. However, it does not produce the best possible fit for a particular input. The network is a mapping that produces an output with a good loss value on average. Unlike the usual case, we can provide all necessary information to evaluate losses at inference time, and so we can polish the network output for a given input. Thus to improve our primitives, we apply a descent procedure on the network predictions using the original loss terms applied to a very large number of samples from the test-time input, refining our predicted primitives significantly.

The number of samples per scene at training time is limited (18K per image); the refinement process allows using a large number of samples (we use 250K) during optimization because the batch size is only 1. We refine for 500 iterations with SGD optimizer, learning rate 0.01, adding about 40 seconds to the inference time.

Furthermore, in our experimentation we have not found existing methods to control the number of primitives particularly effective for this problem [48]. Thus in this work we use simple backward selection to remove primitives. For each primitive, we evaluate the loss with that primitive removed. If the loss does not go up by more than \( \epsilon_{\text{prune}} = 0.001 \) then we remove it and continue searching. In practice, this removal procedure requires \( K \) additional forward passes through the network but helps find a more parsimonious representation of the input.

We remind the reader that the network input is an RGBD image (both training and test time). When applying the optional segmentation loss eqn. 5, a segmentation map is required. At test time, the network still requires an RGBD image to obtain an initial set of primitives; if applying polishing, a depth map is required and segmentation map is optional. At test time, “oracle” refers to GT depth and segmentation being available, whereas “non-oracle” refers to depth and segmentation maps inferred by off-the-shelf depth/segmentation networks.

Finally, before applying this polishing, we can optionally increase the number of primitives by splitting each convex into 8 pieces. For parallelepipeds, the child convexes can inherit the parent’s normals, child offsets are half of the parent’s, and new translations can be computed by taking the midpoint of the original translation coordinate and each of the 8 corners. We focus our experimentation on 0 or 1 splits, as the number of parts grows very quickly. Future work may consider only splitting convexes that show high error with respect to the provided depth or segmentation.

### 3.3. Sampling

The philosophy of our sampling strategy is to encourage placement of convexes to overlap objects in the scene, and discourage placement of convexes where we do not want them. Our losses require samples that can be obtained from the input depth map and camera calibration parameters translating each pixel to a world space 3D coordinate. Given those world space coordinates, we then re-scale the coordinates (per depth map) to lie in a range amenable to training neural networks (approx. a unit cube). Because we work with 240x320 pixel depth maps (4:3 aspect ratio), we rescale X coordinates to lie between \([-\frac{2}{5}, \frac{2}{5}]\); Y coordinates between \([-0.5, 0.5]\) and Z coordinates between \([0,0.8]\). Downstream we need the parameters used to transform a depth map back in the original scale to ray march the predicted decomposition.

**Surface samples** From there we need to generate two classes of samples: “free space” and “surface.” Each sample is a 4-tuple combining the 3D normalized coordinate and a binary label indicating whether the sample is inside the desired volume (1) or outside (0). Inside surface samples are generated by taking the normalized depth coordinates and adding a small constant to Z (we use \( \epsilon_{\text{surf}} = 0.03 \), obtained experimentally) resulting in the tuple \( (X, Y, Z + \epsilon_{\text{surf}}, 1) \). Analogously, outside surface samples are given by \( (X, Y, Z - \epsilon_{\text{surf}}, 0) \).

**Free space samples** We can generate free space samples on the fly as follows. Cast orthographic rays along the Z dimension through each pixel and sample along that ray. If the depth value of the random sample is less than the Z value from the depth at that pixel, that sample is classified as outside. If the depth of the random sample lies between \([Z, Z + t]\) (where \( t \) is a thickness parameter we experimentally set to 0.1) then we classify the sample as inside. The \( Z \)-coordinate of a random free space sample for pixel \((i, j)\) ranges from \([-m_d, Z(i, j) + t]\) where we set \( m_d = 0.1 \) and \( Z(i, j) \) is the re-scaled depth value at pixel \((i, j)\). To avoid convexes deviating far from the inside samples, we
<table>
<thead>
<tr>
<th>Config</th>
<th>Prune</th>
<th>Refine</th>
<th>DepthGT</th>
<th>SegGT</th>
<th>n_start</th>
<th>n_used</th>
<th>AbsRel↓</th>
<th>RMSE↓</th>
<th>Mean↓</th>
<th>Median↓</th>
<th>11.25°↑</th>
<th>22.5°↑</th>
<th>30°↑</th>
<th>SegAcc↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>noInit</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>24</td>
<td>24.0</td>
<td>0.447</td>
<td>1.604</td>
<td>81.286</td>
<td>80.566</td>
<td>0.021</td>
<td>0.061</td>
<td>0.096</td>
<td>0.359</td>
</tr>
<tr>
<td>noSeg</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>24</td>
<td>24.0</td>
<td>0.179</td>
<td>0.664</td>
<td>41.687</td>
<td>38.224</td>
<td>0.115</td>
<td>0.291</td>
<td>0.391</td>
<td>0.626</td>
</tr>
<tr>
<td>withSeg</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>24</td>
<td>24.0</td>
<td>0.310</td>
<td>1.231</td>
<td>32.305</td>
<td>48.846</td>
<td>0.088</td>
<td>0.231</td>
<td>0.317</td>
<td>0.328</td>
</tr>
<tr>
<td>A</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>24</td>
<td>24.0</td>
<td>0.163</td>
<td>0.679</td>
<td>40.692</td>
<td>35.697</td>
<td>0.124</td>
<td>0.313</td>
<td>0.424</td>
<td>0.623</td>
</tr>
<tr>
<td>B</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>24</td>
<td>24.0</td>
<td>0.166</td>
<td>0.696</td>
<td>41.019</td>
<td>35.964</td>
<td>0.122</td>
<td>0.310</td>
<td>0.421</td>
<td>0.623</td>
</tr>
<tr>
<td>C</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>24</td>
<td>14.4</td>
<td>0.144</td>
<td>0.603</td>
<td>38.235</td>
<td>33.621</td>
<td>0.133</td>
<td>0.335</td>
<td>0.451</td>
<td>0.615</td>
</tr>
<tr>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>24</td>
<td>13.9</td>
<td>0.098</td>
<td>0.514</td>
<td>37.355</td>
<td>32.395</td>
<td>0.144</td>
<td>0.335</td>
<td>0.469</td>
<td>0.619</td>
</tr>
<tr>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>24</td>
<td>13.9</td>
<td>0.098</td>
<td>0.514</td>
<td>37.355</td>
<td>32.395</td>
<td>0.144</td>
<td>0.335</td>
<td>0.469</td>
<td>0.618</td>
</tr>
<tr>
<td>ref</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>0.110</td>
<td>0.357</td>
<td>14.9</td>
<td>7.5</td>
<td>0.622</td>
<td>0.793</td>
<td>0.852</td>
<td>0.719</td>
</tr>
</tbody>
</table>

Table 1. Ablation study with different configurations on 654 NYUv2 test images, all error metrics evaluated against ground truth. P refers to whether pruning is applied during refinement, R refers to whether refinement optimization steps are taken, D<sub>GT</sub> and S<sub>SegGT</sub> refer to whether GT depth and segmentation were available during refinement (and inferred by pretrained networks if not). n<sub>start</sub> refers to the number of parts available at training time, n<sub>used</sub> refers to the mean number of convexes remaining post-refinement. Unlike previous work, we can evaluate directly against ground truth depth and normals. We evaluate depth error metrics (AbsRel/RMSE) [39], normal error metrics (subsequent 5 columns) [51], and pixel-wise segmentation accuracy (last column) on the 654 NYUv2 test images. Previous work considers [28, 26]. Segmentation accuracy assumes that we label each convex with the most common label within its boundary from the ground truth segmentation before computing mean pixel-wise accuracy. **noInit** Polishing convexes from a random start point yields very poor results. Since we cannot trust refinement with an arbitrary start point, we train a neural network (on the NYUv2 795 train split). **noSeg** The convexes predicted by a neural network without refinement offer a dramatic improvement over directly optimizing from a random start point. **withSeg** When training with the segmentation loss, all error metrics are worse. We’re not sure why this is, but as we show below, all error metrics significantly improve when we add refinement. **A** We refine the network prediction with all losses except entropy (for segmentation). **B** Same as A, but we refine with the segmentation loss, where segmentation maps are inferred by [6]. Overall, introducing the segmentation loss has a neutral effect across the board. **C** We allow pruning unused convexes in the refinement process via backward selection (note how n<sub>used</sub> drops). **D** Same as A, but allow GT depth maps during refinement instead of inferred. **E** show depth and normal error metrics. **E** We allow GT segmentation maps during refinement instead of inferred. This has an overall neutral effect on the quality as compared with D. **D and E** rely on oracles and the best metrics for oracle systems are italicized. The best non-oracle cfg. was C, and the best metrics are bolded. The last row ref shows the reference error of our pretrained depth estimation network, a recent normal estimation work, and our pretrained segmentation network [39, 1, 6]. Our primitive decompositions using inferred depth/seg in C show depth and segmentation metrics that are only a bit worse than ref., despite the challenging task of representing the scene with just a few primitives. But the normals show a significant gap - this could be due to the presence of curved or complex geometry that can achieve good depth error with just a simple convex, but cannot capture the intricacies in the normals. Further ablations in supplement.

Also construct a shell of "outside" free space samples around all of the aforementioned samples. This shell can be thought of as the difference of two cubes with identical center but different edge length. We let the thickness of this shell be 0.3 and let it start at ±1.2 along the x-direction, ±1.0 along the y-direction, and [−0.5, 1.3] along the z-direction, leaving a min. 0.4 margin between any inside sample and these outside shell samples along all axes. Thus during the training process the convexes have some volume to move freely without any samples encouraging convexes to exist or not exist in this "gap" and they can focus on getting samples near the depth boundary classified correctly.

### 3.4. Learning

As Table 1 indicates, polishing our loss from a random start produces very weak results. Thus training a network to produce a start point is essential.

**Data** Our training procedure requires depth maps with known camera calibration parameters. Like previous scene parsing work, we focus on NYUv2 [34], containing 1449 RGBD images. We use the standard 795-654 train-test split.

While 795 images is usually quite low for training neural networks, we show that the decomposition predicted by the network is actually a good start point for subsequent refinement and achieves a very good fit.

**Network** We use the architecture of [8] consisting of a ResNet18 encoder that accepts an RGBD image and a decoder consisting of 3 fully-connected layers, 1024 neurons each. The output of the decoder consists of parameters representing the translations and blending weight of each convex as well as the normal and offset of each halfplane. The number of halfplanes and convexes is fixed during training time, which determines the number of output parameters.

By forcing each predicted convex to be a parallelepiped, we only need to predict three normals per convex instead of six, which eased training in our experiments.

**Training Details** We implement our method with TensorFlow and train on one NVIDIA A40 GPU. We use Adam Optimizer and learning rate 0.0001. We set our minibatch size to 32 images (requiring 2.5 hours for 20k iterations) and randomly sample 6k free space and 12k surface samples per image in each element of the minibatch. 10% of the
Figure 5. Qualitative evaluation of our method on depth. These are random test images not seen during the training process. The third row shows having only inferred depth and segmentation during refinement, whereas the last row shows having GT depth/seg available during refinement. AbsRel depth superimposed over predictions. Fairly good approximations of depth can be generated with just a few convexes - 24 at train time, and about 14 remain after refinement. Only a small penalty in quality is observed by not having GT depth/seg available.

free space samples are taken from the outside shell. Within the sample loss, we also observed slightly better quality and more stable training by annealing the relative weight of free space samples vs. surface samples, starting from a 0.9 : 0.1 split and saturating to 0.5 : 0.5 midway through training. Intuitively, this allows the network to focus on big picture features of the geometric layout early on, and then adjust fine-scale details of the surface near the end.

**4. Experiments & Evaluation**

We investigate several design choices in our pipeline. The primary finding is that our network alone (regression) produces reasonable but not great convex decompositions. Polishing alone (descent) generates completely unusable primitives. But allowing network input as initialization of our refinement procedure results in our best quality. Our results remain strong even when ground truth depths and segmentations are not available during refinement and must be inferred by other networks. In the remainder of this section, we show that our mixed procedure produces good depth, normals, and segmentation on our test dataset.

**Quantitative evaluation of normals and depth**: Given parameters of a collection of convexes for a given test image, we can ray march from the original viewpoint to obtain a depth map, part segmentation, and normals. Our method involves ray marching and interval halving at the first inter-

Figure 6. Qualitative evaluation of our method on normals. These are random test images not seen during the training process. The third row shows having only inferred depth and segmentation during refinement, whereas the last row shows having GT depth/seg available during refinement. Mean/median normal error in degrees superimposed over predictions. Fairly good approximations of normals can be generated with just a few convexes - 24 at train time, and about 14 remain after refinement.

Figure 7. Qualitative evaluation of our method on segmentation. These are random test images not seen during the training process. The third row shows having only inferred depth and segmentation during refinement, whereas the last row shows having GT depth/seg available during refinement. Mean per-pixel segmentation accuracy is superimposed over predictions. Fairly good approximations of segmentation can be generated with just a few convexes - 24 at train time, and about 14 remain after refinement.

section point. In Table 1 we use the standard depth error metrics to evaluate the depth associated with the convex decomposition against the input depth map, evaluating on the 654 NYUv2 test images from [28]. Although we train with
Random Start

The final normal per pixel is given by normalizing per-pixel 3D coordinates \( (X_{ij}, Y_{ij}, Z_{ij}) \). We can compute the normals from the input depth map for evaluating normals of the predicted convexes. To obtain normals from the collection of convexes, we transform the predicted depth map back to world space coordinates when computing depth error metrics. The final normal per pixel is given by normalizing: \( (\frac{\nabla X}{\sqrt{\nabla X^2 + \nabla Y^2}}, 1) \).

To obtain normals from the collection of convexes, we can compute the gradient of the implicit function that defines it and evaluate it per-pixel at the first ray-surface intersection point for each pixel:

\[
\nabla \Phi(x) = \frac{\delta \sum_h n_h e^\delta(n_h \cdot x + d_h)}{\sum_h e^\delta(n_h \cdot x + d_h)}
\]

Note that sums over \( h \) iterate over all the halfplanes associated with the convex that the ray hit first.

Our error metrics strongly indicate that polishing alone produces unusable results, network alone produces better results that often resemble the input, and our very best results come from a mixed method that uses the network prediction as a start point for subsequent refining. Our pruning procedure shows that we can improve depth and normal error metrics while reducing the number of parts. However, our ablations showed that our entropy-based segmentation loss yielded a neutral effect on the segmentation quality. That loss did help in toy models we tried, so we leave it in the paper to possibly help future research.

We compute the Chamfer-L1 distance of the predicted and ground truth depth map (in meters). Our best non-oracle network (row C, Table 1) got a score of 1.46; our best oracle network (row E, Table 1) got a score of 0.54.

Quantitative evaluation of segmentation: Ideally, primitives “explain” objects, so each primitive should span only one object (though individual objects could consist of more than one primitive). We can evaluate this property by comparing primitive based segmentations with ground truth for the test partition of NYUv2. We render an image where each pixel has the number of the visible primitive at that pixel. Each primitive then gets the most common label in its support. Finally, we compute the pixel accuracy of this labeling. Note that this test is unreliable if there are many primitives (for example, one per pixel), but we have relatively few. A high pixel accuracy rate strongly suggests that each primitive spans a single object. Our mean pixel accuracy rate for different configurations is shown in Table 1, last column. Our best configurations, C & D, showed a segmentation accuracy rate of 0.618. We also show qualitative results in Fig. 7. We conclude that our primitive decomposition mostly identifies objects, but could likely be improved by having more primitives.

Qualitative evaluation We compare our method against previous work [26] on six NYUv2 images in Fig. 4. Our representation generally covers the whole depth map with-
out leaving gaps, and finds more convexes that correspond with real-world objects. We also evaluate our method specifically looking at depth (Fig. 5), normals (Fig. 6), and segmentation (Fig. 7). Our network-initialized refinement procedure accurately predicts convexes whose depth and normals closely match the ground truth. This is true even when GT depth isn’t available to the refinement. Finally, we evaluate whether our hybrid method is necessary in Fig. 8, showing that polishing without a network (random start) does not produce useful results due to the extremely difficult optimization problem; fair quality with running our network but no polishing; and our very best quality when running our network followed by polishing (hybrid method).

5. Conclusion

We presented a novel procedure to decompose complex indoor scenes into 3D primitives, relying on both regression and descent. By capturing the whole input, we used existing depth, normal, and segmentation error metrics to evaluate, unlike previous work. Further, our qualitative evaluation shows that our primitives map to real-world objects quite well, capturing more of the input than previous work. By relying on suitable pretrained networks, our experiments showed successful primitive decomposition when ground truth depth or segmentation isn’t available.

One application we have had success with is generating scenes conditioned on the primitives - we can edit the geometry of the scene by simply moving a cuboid, far easier than editing say a depth map [50]. Future work may analyze the latent space, perhaps with applications to generative modeling or scene graph construction. Training with a large scale RGBD+Segmentation dataset could also improve the start points of our method. Further, a natural extension is generating primitives that are temporally consistent across video frames.

Our refinement process showed it’s possible to control the number of primitives and their granularity even though the start point was produced by a network trained with a fixed number of convexes. Some scenes could benefit from a convex merge process such as flat walls that are decomposed into several convexes. Such a procedure may improve qualitative results when we train with more convexes than the input requires.

6. Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 2106825 and by gifts from Amazon and Boeing.

References


