CvxNet:
Learnable Convex Decomposition

Boyang Deng
Google Research
Kyle Genova
Google Research
Soroosh Yazdani
Google Hardware
Sofien Bouaziz
Google Hardware
Geoffrey Hinton
Google Research
Andrea Tagliasacchi
Google Research

Abstract

Any solid object can be decomposed into a collection of convex polytopes (in short, convexes). When a small number of convexes are used, such a decomposition can be thought of as a piece-wise approximation of the geometry. This decomposition is fundamental in computer graphics, where it provides one of the most common ways to approximate geometry, for example, in real-time physics simulation. A convex object also has the property of being simultaneously an explicit and implicit representation: one can interpret it explicitly as a mesh derived by computing the vertices of a convex hull, or implicitly as the collection of half-space constraints or support functions. Their implicit representation makes them particularly well suited for neural network training, as they abstract away from the topology of the geometry they need to represent. However, at testing time, convexes can also generate explicit representations – polygonal meshes – which can then be used in any downstream application. We introduce a network architecture to represent a low dimensional family of convexes. This family is automatically derived via an auto-encoding process. We investigate the applications of this architecture including automatic convex decomposition, image to 3D reconstruction, and part-based shape retrieval.

1. Introduction

While images admit a standard representation in the form of a scalar function uniformly discretized on a grid, the curse of dimensionality has prevented the effective usage of analogous representations for learning 3D geometry. Voxel representations have shown some promise at low resolution [10, 20, 35, 57, 62, 69, 74], while hierarchical representations have attempted to reduce the memory footprint required for training [58, 64, 73], but at the significant cost of complex implementations. Rather than representing the volume occupied by a 3D object, one can resort to modeling its surface via a collection of points [1, 19], polygons [31, 56, 71], or surface patches [26]. Alternatively, one might follow Cezanne’s advice and “treat nature by means of the cylinder, the sphere, the cone, everything brought into proper perspective”, and think to approximate 3D geometry as geons [4] – collections of simple to interpret geometric primitives [68, 77], and their composition [60, 21]. Hence, one might rightfully start wondering “why so many representations of 3D data exist, and why would one be more advantageous than the other?” One observation is that multiple equivalent representations of 3D geometry exist because real-world applications need to perform different operations and queries on this data ( [9, Ch.1]). For example, in computer graphics, points and polygons allow for very efficient rendering on GPUs, while volumes allow artists to sculpt geometry without having to worry about tes-
sellation [51] or assembling geometry by smooth composition [2], while primitives enable highly efficient collision detection [66] and resolution [67]. In computer vision and robotics, analogous trade-offs exist: surface models are essential for the construction of low-dimensional parametric templates essential for tracking [6, 8], volumetric representations are key to capturing 3D data whose topology is unknown [48, 47], while part-based models provide a natural decomposition of an object into its semantic components. Part-based models create a representation useful to reason about extent, mass, contact, ... quantities that are key to describing the scene, and planning motions [29, 28].

Contributions. In this paper, we propose a novel representation for geometry based on primitive decomposition. The representation is parsimonious, as we approximate geometry via a small number of convex elements, while we seek to allow low-dimensional representation to be automatically inferred from data – without any human supervision. More specifically, inspired by recent works [68, 21, 44] we train our pipeline in an unsupervised manner: predicting the primitive configuration as well as their parameters by checking whether the reconstructed geometry matches the geometry of the target. We note how we inherit a number of interesting properties from several of the aforementioned representations. As it is part-based it is naturally locally supported, and by training on a shape collection, parts have a semantic association (i.e. the same element is used to represent the backs of chairs). Although part-based, each of them is not restricted to belong to the class of boxes [68], ellipsoids [21], or sphere-meshes [67], but to the more general class of convexes. As a convex is defined by a collection of half-space constraints, it can be simultaneously decoded into an explicit (polygonal mesh), as well as implicit (indicator function) representation. Because our encoder decomposes geometry into convexes, it is immediately usable in any application requiring real-time physics simulation, as collision resolution between convexes is efficiently decided by GJK [23] (Figure 1). Finally, parts can interact via structuring [21] to generate smooth blending between parts.

2. Related works

One of the simplest high-dimensional representations is voxels, and they are the most commonly used representation for discriminative [43, 54, 61] models, due to their similarity to image based convolutions. Voxel have also been used successfully for generative models [75, 16, 24, 57, 62, 74]. However, the memory requirements of voxels makes them unsuitable for resolutions larger than 64³. One can reduce the memory consumption significantly by using octrees that take advantage of the sparsity of voxels [58, 72, 73, 64]. This can extend the resolution to 512³, for instance, but comes at the cost of more complicated implementation.

Surfaces. In computer graphics, polygonal meshes are the standard representation of 3D objects. Meshes have also been considered for discriminative classification by applying graph convolutions to the mesh [42, 11, 27, 46]. Recently, meshes have also been considered as the output of a network [26, 32, 71]. A key weakness of these models is the fact that they may produce self-intersecting meshes. Another natural high-dimensional representation that has garnered some traction in vision is the point cloud representation. Point clouds are the natural representation of objects if one is using sensors such as depth cameras or LiDAR, and they require far less memory than voxels. Qi et al. [53, 55] used point clouds as a representation for discriminative deep learning tasks. Hoppe et al. [30] used point clouds for surface mesh reconstruction (see also [3] for a survey of other techniques). Fan et. al. [19] and Lin et. al. [37] used point clouds for 3D reconstruction using deep learning. However, these approaches require additional non-trivial post-processing steps to generate the final 3D mesh.

Primitives. Far more common is to approximate the input shape by set of volumetric primitives. With this perspective in mind, representing shapes as voxels will be a special case, where the primitives are unit cubes in a lattice. Another fundamental way to describe 3D shapes is via Constructive Solid Geometry [33]. Sherma et. al. [60] presents a model that will output a program (i.e. set of Boolean operations on shape primitives) that generate the input image or shape. In general, this is a fairly difficult task. Some of the classical primitives used in graphics and computer vision are blocks world [59], generalized cylinders [5], geons [4], and even Lego pieces [70]. In [68], a deep CNN is used to interpret a shape as a union of simple rectangular prisms. They also note that their model provides a consistent parsing across shapes (i.e. the head is captured by the same primitive), allowing some interpretability of the output. In [50], they extended cuboids to superquadrics, showing that the extra flexibility will result in better reconstructions.

Implicit surfaces. If one generalizes the shape primitives to analytic surfaces (i.e. level sets of analytic functions), then new analytic tools become available for generating shapes. In [44, 15], for instance, they train a model to discriminate inside coordinates from outside coordinates (referred to as an occupancy function in the paper, and as an indicator function in the graphics community). Park et. al. [49] used the signed distance function to the surface of the shape to achieve the same goal. One disadvantage of the implicit description of the shape is that most of the interpretability is missing from the final answer. In [21], they take a more geometric approach and restrict to level sets of axis-aligned Gaussians. Partly due to the restrictions of these functions, their representation struggles on shapes with angled parts, but they do recover the interpretability that [68] offers.
Convex decomposition. In graphics, a common method to represent shapes is to describe them as a collection of convex objects. Several methods for convex decomposition of meshes have been proposed [25, 52]. In machine learning, however, we only find early attempts to approach convex hull computation via neural networks [34]. Splitting the meshes into exactly convexes generally produces too many pieces [13]. As such, it is more prudent to seek small number of convexes that approximate the input shape [22, 36, 38, 41, 40]. Recently [66] also extended convex decomposition to the spatio-temporal domain, by considering moving geometry. Our method is most related to [68] and [21], in that we train an occupancy function. However, we choose our space of functions so that their level sets are approximately convex, and use these as building blocks.

3. Method – CvxNet

Our object is represented via an indicator function $O : \mathbb{R}^3 \rightarrow [0, 1]$, and with $\partial O = \{x \in \mathbb{R}^3 \mid O(x) = 0.5\}$ we indicate the surface of the object. The indicator function is defined such that $\{x \in \mathbb{R}^3 \mid O(x) = 0\}$ defines the outside of the object and $\{x \in \mathbb{R}^3 \mid O(x) = 1\}$ the inside. Given an input (e.g. an image, point cloud, or voxel grid) an encoder estimates the parameters $\{\beta_k\}$ of our template representation $O(\cdot)$ with $K$ primitives (indexed by $k$). We then evaluate the template at random sample points $x$, and our training loss ensures $\hat{O}(x) \approx O(x)$. In the discussion below, without loss of generality, we use 2D illustrative examples where $O : \mathbb{R}^2 \rightarrow [0, 1]$. Our representation is a differentiable convex decomposition, which is used to train an image encoder in an end-to-end fashion. We begin by describing a differentiable representation of a single convex object (Section 3.1). Then we introduce an auto-encoder architecture to create a low-dimensional family of approximate convexes (Section 3.2). These allow us to represent objects as spatial compositions of convexes (Section 3.4). We then describe the losses used to train our networks (Section 3.5) and mention a few implementation details (Section 3.6).

3.1. Differentiable convex indicator – Figure 2

We define a decoder that given a collection of (unordered) half-space constraints constructs the indicator function of a single convex object; such a function can be evaluated at any point $x \in \mathbb{R}^3$. We define $\mathcal{H}_h(x) = n_h \cdot x + d_h$ as the signed distance of the point $x$ from the $h$-th plane with normal $n_h$ and offset $d_h$. Given a sufficiently large number $H$ of half-planes the signed distance function of any convex object can be approximated by taking the max of the signed distance functions of the planes. To facilitate gradient learning, instead of maximum, we use the smooth maximum function LogSumExp and define the approximate signed distance function, $\Phi(x)$:

$$\Phi(x) = \text{LogSumExp}(\delta \mathcal{H}_h(x)),$$

(1)

Note this is an approximate SDF, as the property $\|\nabla \Phi(x)\| = 1$ is not necessarily satisfied $\forall x$. We then convert the signed distance function to an indicator function $C : \mathbb{R}^3 \rightarrow [0, 1]$:

$$C(x|\beta) = \text{Sigmoid}(-\sigma \Phi(x)),$$

(2)

We denote the collection of hyperplane parameters as $h = \{\{n_h, d_h\}\}$, and the overall set of parameters for a convex as $\beta = [h, \sigma]$. We treat $\sigma$ as a hyperparameter, and consider the rest as the learnable parameters of our representation. As illustrated in Figure 2, the parameter $\delta$ controls the smoothness of the generated convex, while $\sigma$ controls the sharpness of the transition of the indicator function. Similar to the smooth maximum function, the soft classification boundary created by Sigmoid facilitates training.

In summary, given a collection of hyperplane parameters, this differentiable module generates a function that can be evaluated at any position $x$.

3.2. Convex encoder/decoder – Figure 3

A sufficiently large set of hyperplanes can represent any convex object, but one may ask whether it would be pos-
Convex auto-encoder – The encoder $E$ creates a low dimensional latent vector representation $\lambda$, decoded into a collection of hyperplanes by the decoder $D$. The training loss involves reconstructing the value of the input image at random pixels $x$.

It is possible to discover some form of correlation between their parameters. Towards this goal, we employ an auto-encoder architecture illustrated in Figure 3. Given an input, the encoder $E$ derives a bottleneck representation $\lambda$ from the input. Then, a decoder $D$ derives the collection of hyperplane parameters. While in theory permuting the $H$ hyperplanes generates the same convex, the decoder $D$ correlates a particular hyperplane with a corresponding orientation. This is visible in Figure 4, where we color-code different 2D hyperplanes and indicate their orientation distribution in a simple 2D auto-encoding task for a collection of axis-aligned ellipsoids. As ellipsoids and oriented cuboids are convexes, we argue that the architecture in Figure 3 allows us to generalize the core geometric primitives proposed in VP [68] and SIF [21]; we verify this claim in Figure 5.

Correlation – While the description of a convex, $\{(n_h, d_h)\}$, is permutation invariant we employ an encoder/decoder that implicitly establishes an ordering. Our visualization reveals how a particular hyperplane typically represents a particular subset of orientations.

Interpolation – We compute latent code of shapes in the corners using CvxNet. We then linearly interpolate latent codes to synthesize shapes in-between. Our primitives generalize the shape space of VP [68] (boxes) and SIF [21] (ellipsoids) so we can interpolate between them smoothly.

3.3. Explicit interpretation – Figure 6

What is significantly different from other methods that employ indicator functions as trainable representations of 3D geometry, is that convexes generated by our network admit an explicit interpretation: they can be easily converted into polygonal meshes. This is in striking contrast to [49, 15, 21, 44], where a computationally intensive isosurfacing operation needs to be executed to extract their surface (e.g. Marching Cubes [39]). More specifically, isosurfacing techniques typically suffer the curse of dimensionality, with a performance that scales as $1/\varepsilon^d$, where $\varepsilon$ the desired spatial resolution and $d$ is usually 3. Conversely, as we illustrate in Figure 6, we only require the execution of two duality transforms, and the computations of two convex hulls of $H$ points. The complexity of these operations is clearly independent of any resolution parameter $\varepsilon$.

3.4. Multi convex decomposition – Figure 7

Having a learnable pipeline for a single convex object, we can now expand the expressivity of our model by representing generic non-convex objects as compositions of convexes [66]. To achieve this task an encoder $E$ outputs a low-

Figure 3.

Figure 4.

Figure 5.

Figure 6.

Figure 7.
Multi-convex auto-encoder – Our network approximates input geometry as a composition of convex elements. Note that this network does not prescribe how the final image is generated, but merely output the shape \( \{ \beta_k \} \) and pose \( \{ T_k \} \) parameters of the abstraction. Note that this is an illustration where the parameters \( \{ \beta_k \}, \{ T_k \} \) have been directly optimized via SGD with a preset \( \delta \).

dimensional bottleneck representation of all \( K \) convexes \( \lambda \) that \( D \) decodes into a collection of \( K \) parameter tuples. Each tuple (indexed by \( k \)) is comprised of a shape code \( \beta_k \), and corresponding transformation \( T_k(x) = x + c_k \) that transforms the point from world coordinates to local coordinates. \( c_k \) is the predicted translation vector (Figure 7).

3.5. Training losses

First and foremost, we want the (ground truth) indicator function of our object \( O \) to be well approximated:

\[
L_{\text{approx}}(\omega) = \mathbb{E}_{x \sim \mathbb{R}^3} \| \hat{O}(x) - O(x) \| ^2 ,
\]

where \( \hat{O}(x) = \max_k \{ C_k(x) \} \), and \( C_k(x) = C(T_k(x)|\beta_k) \). The application of the \( \max \) operator produces a perfect union of convexes. While constructive solid geometry typically applies the \( \min \) operator to compute the union of signed distance functions, note that we apply the \( \max \) operator to indicator functions instead with the same effect; see Section 6 in the supplementary material for more details. We couple the approximation loss with several auxiliary losses that enforce the desired properties of our decomposition.

Decomposition loss (auxiliary). We seek a parsimonious decomposition of an object akin to Tulsiani et al. [68]. Hence, overlap between elements should be discouraged:

\[
L_{\text{decomp}}(\omega) = \mathbb{E}_{x \sim \mathbb{R}^3} \| \text{relu} \left( \sum_k \{ C_k(x) \} - \tau \right) \| ^2 ,
\]

where we use a permissive \( \tau = 2 \), and note how the ReLU activates the loss only when an overlap occurs.

Unique parameterization loss (auxiliary). While each convex is parameterized with respect to the origin, there is a nullspace of solutions – we can move the origin to another location within the convex, and update offsets \( \{ d_h \} \) and transformation \( T \) accordingly – see Figure 8(left). To remove such a null-space, we simply regularize the magnitudes of the offsets for each of the \( K \) elements:

\[
L_{\text{unique}}(\omega) = \frac{1}{H} \sum_h \| d_h \| ^2
\]

In the supplementary material, we prove that minimizing \( L_{\text{unique}} \) leads to a unique solution and centers the convex body to the origin. This loss further ensures that “inactive” hyperplane constraints can be readily re-activated during learning. Because they fit tightly around the surface they are therefore sensitive to shape changes.

Guidance loss (auxiliary). As we will describe in Section 3.6, we use offline sampling to speed-up training. However, this can cause severe issues. In particular, when a convex “falls within the cracks” of sampling (i.e. \( \exists x \mid C(x) > 0.5 \)), it can be effectively removed from the learning process. This can easily happen when the convex enters a degenerate state (i.e. \( d_h = 0 \ \forall h \)). Unfortunately these degenerate configurations are encouraged by the loss (5). We can prevent collapses by ensuring that each of them represents

Figure 7. Multi-convex auto-encoder – Our network approximates input geometry as a composition of convex elements. Note that this network does not prescribe how the final image is generated, but merely output the shape \( \{ \beta_k \} \) and pose \( \{ T_k \} \) parameters of the abstraction. Note that this is an illustration where the parameters \( \{ \beta_k \}, \{ T_k \} \) have been directly optimized via SGD with a preset \( \delta \).

Figure 8. Auxiliary losses – Our \( L_{\text{unique}} \) loss (left) prevents the existence of a null-space in the specification of convexes, and (middle) ensures inactive hyperplanes can be easily activated during training. (right) Our \( L_{\text{guide}} \) move convexes towards the representation of samples drawn from within the object \( x \in O \).
and hyperplanes. For the {Depth}-to-3D task, we use 50 
convexes each with 25 hyperplanes. Similar to OccNet [44], we use ResNet18 as the encoder $E$ for both the {Depth}-to-3D and the RGB-to-3D experiments. A fully connected layer then generates the latent code $\lambda \in \mathbb{R}^{256}$ that is provided as input to the decoder $D$. For the decoder $D$ we use a sequential model with four hidden layers with $(1024, 1024, 2048, |\mathbf{H}|)$ units respectively. The output dimension is $|\mathbf{H}| = K(4 + 3H)$ where for each of the $K$ elements we specify a translation (3 DOFs) and a smoothness (1 DOFs). Each hyperplane is specified by the (unit) normal and the offset from the origin (3H DOFs). In all our experiments, we use a batch of size 32 and train with Adam with a learning rate of $10^{-4}$, $\beta_1 = .9$, and $\beta_2 = .999$. As determined by grid-search on the validation set, we set the weight for our losses $\{L_{\text{approx}} : 1.0, L_{\text{decomp}} : 0.1, L_{\text{unique}} : 0.001, L_{\text{guide}} : 0.01, L_{\text{loc}} : 1.0\}$ and $\sigma = 75$.

4. Experiments

We use the ShapeNet [12] dataset in our experiments. We use the same voxelization, renderings, and data split as in Choy et. al. [16]. Moreover, we use the same multi-view depth renderings as [21] for our {Depth}-to-3D experiments, where we render each example from cameras re-arranged on the vertices of a dodecahedron. Note that this problem is a harder problem than 3D auto-encoding with point cloud input as proposed by OccNet [44] and resembles more closely the single view reconstruction problem. At training time we need ground truth inside/outside labels, so we employ the watertight meshes from [44] – this also ensures a fair comparison to this method. For the quantitative evaluation of semantic decomposition, we use labels from PartNet [45] and exploit the overlap with ShapeNet.

Methods. We quantitatively compare our method to a number of self-supervised algorithms with different characteristics. First, we consider VP [68] that learns a parsimonious approximation of the input via (the union of) oriented boxes. We also compare to the Structured Implicit Function SIF [21] method that represents solid geometry as an iso-level of a sum of weighted Gaussians; like VP [68], and in contrast to OccNet [44], this method provides an interpretable encoding of geometry. Finally, from the class of techniques that directly learn non-interpretative representations of implicit functions, we select OccNet [44], P2M [71], and AtlasNet [26]; in contrast to the previous methods, these solutions do not provide any form of shape decomposition. As OccNet [44] only report results on RGB-to-3D tasks, we extend the original codebase to also solve {Depth}-to-3D tasks. We follow the same data pre-processing used by SIF [21].

Metrics. With $\hat{O}$ and $\partial \hat{O}$ we respectively indicate the indicator and the surface of the union of our primitives. We then
use three quantitative metrics to evaluate the performance of 3D reconstruction: ① The Volumetric IoU; note that with 100K uniform samples to estimate this metric, our estimation is more accurate than the 323 voxel grid estimation used by [16]. ② The Chamfer-L1 distance, a smooth relaxation of the symmetric Hausdorff distance measuring the average between reconstruction accuracy and completeness [16]. ③ Following the arguments presented in [65], we also employ F-score to quantitatively assess performance. It can be understood as “the percentage of correctly reconstructed surface”.

4.1. Abstraction – Figure 9, 10, 11

As our convex decomposition is learnt on a shape collection, the convexes produced by our decoder are in natural correspondence – e.g. we expect the same k-th convex to represent the leg of a chair in the chairs dataset. We analyze this quantitatively on the PartNet dataset [45]. We do so by verifying whether the k-th component is consistently mapped to the same PartNet part label; see Figure 11 (left) for the distribution of PartNet labels within each component. We can then assign the most commonly associated label to a given convex to segment the PartNet point cloud, achieving a relatively high accuracy; see Figure 11 (right). This reveals how our representation captures the semantic structure in the dataset. We also evaluate our shape abstraction capabilities by varying the number of components and evaluating the trade-off between representation parsimony and reconstruction accuracy; we visualize this via Pareto-optimal curves in the plot of Figure 9. We compare with SIF [21], and note that thanks to the generalized shape space of our model, our curve dominates theirs regardless of the number of primitives chosen. We further investigate the use of natural correspondence in a part-based retrieval task. We first encode an input into our representation, allow a user to select a few parts of interest, and then use this (incomplete) shape-code to fetch the elements in the training set with the closest (partial) shape-code; see Figure 10.

4.2. Reconstruction – Table 1 and Figure 12

We quantitatively evaluate the reconstruction performance against a number of state-of-the-art methods given inputs as multiple depth map images (Depth-to-3D) and a single color image (RGB-to-3D); see Table 1. A few qualitative examples are displayed in Figure 12. We find that CvxNet is: ① consistently better than other decomposition methods (SIF, VP, and SQ) which share the common goal of learning shape elements; ② in general comparable to the state-of-the-art reconstruction methods; ③ better than the leading technique (OccNet [44]) when evaluated in terms of F-score, and tested on multi-view depth input.
Table 1. Reconstruction performance on ShapeNet/Multi – We evaluate our method against P2M [71], AtlasNet [26], OccNet [44] and SIF [21]. We provide in input either (left) a collection of depth maps or (right) a single color image. For AtlasNet [26], note that IoU cannot be measured as the meshes are not watertight. We omit VP [68], as it only produces a very rough shape decomposition.

4.3. Ablation studies

We summarize here the results of several ablation studies found in the supplementary material. Our analysis reveals that the method is relatively insensitive to the dimensionality of the bottleneck \( \lambda \). We also investigate the effect of varying the number of convexes \( K \) and number of hyperplanes \( H \) in terms of reconstruction accuracy and inference/training time. Moreover, we quantitatively demonstrate that using signed distance as supervision for \( L_{\text{approx}} \) produces significantly worse results and at the cost of slightly worse performance we can collapse \( L_{\text{guide}} \) and \( L_{\text{loc}} \) into one. Finally, we perform an ablation study with respect to our losses, and verify that each is beneficial towards effective learning.

5. Conclusions

We propose a differentiable representation of convex primitives that is amenable to learning. The inferred representations are directly usable in graphics/physics pipelines; see Figure 1. Our self-supervised technique provides more detailed reconstructions than very recently proposed part-based techniques (SIF [21] in Figure 9), and even consistently outperforms the leading reconstruction technique on multi-view input (OccNet [44] in Table 1). In the future we would like to generalize the model to be able to predict a variable number of parts [68], understand symmetries and modeling hierarchies [76], and include the modeling of rotations [68]. Leveraging the invariance of hyperplane ordering, it would be interesting to investigate the effect of permutation-invariant encoders [63], or remove encoders altogether in favor of auto-decoder architectures [49].

Acknowledgements. We would like to acknowledge Luca Prasso and Timothy Jeruzalski for their help with preparing the rigid-body simulations, Avneesh Sud and Ke Li for reviewing our draft, and Anton Mikhailov, Tom Funkhouser, and Erwin Coumans for fruitful discussions.
References


[71] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European Conference on Computer Vision*, 2018. 1, 2, 6, 8


CvxNet: Learnable Convex Decomposition
Supplementary Material

6. Union of smooth indicator functions

We define the smooth indicator function for the $k$-th object:

$$C_k(x) = \text{Sigmoid}_p(-\Phi_k^p(x)),$$ \hspace{1cm} (8)

where $\Phi_k^p(x)$ is the $k$-th object signed distance function. In constructive solid geometry the union of signed distance function is defined using the min operator. Therefore the union operator for our indicator function can be written:

$$U\{C_k(x)\} = \text{Sigmoid}_p(-\min_k\{\Phi_k^p(x)\})$$ \hspace{1cm} (9)

$$= \text{Sigmoid}_p(\max_k\{-\Phi_k^p(x)\})$$ \hspace{1cm} (10)

$$= \max_k\{\text{Sigmoid}_p(-\Phi_k^p(x))\} = \max_k\{C_k(x)\}.$$

Note that the max operator is commutative with respect to monotonically increasing functions allowing us to extract the max operator from the Sigmoid$_p(\cdot)$ function in (10).

Figure 13. Depth vs. Color – A qualitative example illustrating the degradation in performance as we move from {depth} to a “weaker” RGB input. In the top row we illustrate a model where the frequency of the surface is low. In this case, both {depth} input and RGB input can approximate the shape relatively accurately. In contrast, the bottom row shows when the ground truth surface has high frequency, the RGB input results lose many details while the {depth} input results remain accurate.

Figure 14. Original Losses vs. Merged Losses – A quantitative comparison between the original losses and the merged version where guidance loss and localization loss are collapsed as described in 7. At the cost of slightly worse performance, we can simplify the training objective.

<table>
<thead>
<tr>
<th>Losses</th>
<th>IoU</th>
<th>Chamfer-$L_1$</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>0.731</td>
<td>0.080</td>
<td>73.49%</td>
</tr>
<tr>
<td>Merged</td>
<td>0.720</td>
<td>0.087</td>
<td>71.62%</td>
</tr>
</tbody>
</table>

Figure 15. Single-class vs. multi-class – It is interesting to note that training on single vs. multi class has a behavior different from what one would expect (i.e. overfitting to a single class is beneficial). Note how training in multi-class on the {Depth}-to-3D input improves the reconstruction performance across the entire benchmark. Conversely, with RGB-to-3D input, single class training is beneficial in most cases. We explain this by the fact that RGB inputs have complex texture which is stable within each class but not easily transferable across classes. Contrarily, local geometries learned from {depth} are agnostic to classes.

<table>
<thead>
<tr>
<th>H</th>
<th>12</th>
<th>25</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0.709 / 0.139 / 44</td>
<td>0.712 / 0.127 / 59</td>
<td>0.714 / 0.124 / 87</td>
</tr>
<tr>
<td>25</td>
<td>0.717 / 0.100 / 60</td>
<td>0.720 / 0.099 / 92</td>
<td>0.721 / 0.096 / 153</td>
</tr>
<tr>
<td>50</td>
<td>0.724 / 0.088 / 92</td>
<td>0.730 / 0.083 / 156</td>
<td>0.731 / 0.080 / 280</td>
</tr>
</tbody>
</table>

Figure 16. Ablation on model complexity – We analyze how the number of hyperplanes $H$ and number of convexes $K$ relate to mIoU / Chamfer-$L_1$ / Inference Time (ms). The mIoU and Chamfer-$L_1$ are measured on the test set of ShapeNet (multi-class) with multi-view depth input. We measure the inference time of a batch with 32 examples and 2048 points, which is equivalent to one forward propagation step at training time.

7. Merged guidance loss and localization loss

While the guidance loss (6) and localization loss (7) are designed by different motivations, they are inherently consistent in encouraging the convex elements to reside close to the ground truth. Therefore, we propose an alternative train-
### Table 1: Ablation on losses – We test on ShapeNet multi with RGB input, where each column (from 2 to 5) removes (−) one loss term from the training. We observe that each loss can improve the overall performance.

<table>
<thead>
<tr>
<th>Metric</th>
<th>-L_decomp</th>
<th>-L_unique</th>
<th>-L_guide</th>
<th>-L_loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vol-IoU</td>
<td>0.567</td>
<td>0.558</td>
<td>0.545</td>
<td>0.551</td>
</tr>
<tr>
<td>Chamfer</td>
<td>0.245</td>
<td>0.308</td>
<td>0.313</td>
<td>0.335</td>
</tr>
<tr>
<td>F%</td>
<td>47.36</td>
<td>45.29</td>
<td>44.03</td>
<td>46.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 17. Ablation on losses – We test on ShapeNet multi with RGB input, where each column (from 2 to 5) removes (−) one loss term from the training. We observe that each loss can improve the overall performance.

### Table 2: Supervision vs. Indicator Training – We study the difference between learning signed distance functions (i.e., replacing $O(x)$ with signed distance in (3) and removing Sigmoid in (2)) and learning indicator functions. Note how the performance would degenerate significantly when the model learns to predict signed distances.

<table>
<thead>
<tr>
<th>Supervision</th>
<th>IoU</th>
<th>Chamfer-L1</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicator</td>
<td>0.747</td>
<td>0.035</td>
<td>83.22%</td>
</tr>
<tr>
<td>SDF</td>
<td>0.650</td>
<td>0.045</td>
<td>73.08%</td>
</tr>
</tbody>
</table>

Figure 18. Ablation SDF Training vs. Indicator Training – We study the difference between learning signed distance functions (i.e., replacing $O(x)$ with signed distance in (3) and removing Sigmoid in (2)) and learning indicator functions. Note how the performance would degenerate significantly when the model learns to predict signed distances.

### Table 3: Ablation latent size – We study the performance of our architecture as we vary the number of latent dimensions in Figure 3. Note how the reconstruction performance remains relatively stable as we vary this hyper-parameter.

| $|\lambda|$ | F-Score |
|----------|---------|
| 32       | 72.24%  |
| 64       | 73.31%  |
| 128      | 73.14%  |
| 256      | 73.49%  |

Figure 19. Ablation latent size – We study the performance of our architecture as we vary the number of latent dimensions in Figure 3. Note how the reconstruction performance remains relatively stable as we vary this hyper-parameter.

### Figure 20: Shape interpolation – Analogously to Figure 5, we encode the four models in the corners, and generate the intermediate results via linear interpolation. These results can be better appreciated in the supplementary video where we densely interpolate between the corner examples.

The idea is that while $\Phi_k(x)$ is approximate, it can still be used to get weak gradients. Essentially, this means that each convex needs to “explain” the N closest samples of $O$.

### 8. Proof of auxiliary null-space loss

Consider a collection of hyperplanes $h = \{(n_h, d_h)\}$ where each pair $(n_h, d_h)$ denotes $H_h(x) = n_h \cdot x + d_h$. We like to show that amongst all possible translations of this collection, there is a unique one that minimizes $L_{\text{unique}} = \frac{1}{H} \sum_h H_h(x)^2$. We can rewrite this sum as

$$||(H_h(x))_h||^2 = ||(d_h + n_h \cdot x)_h||^2 = ||d_h + N x||^2$$

where $d_h = (d_h)_h$ is a vector with entries $d_h$, and $N$ is the matrix formed by $n_h$ as columns. However it is well known that the above minimization has a unique solution, and the minimizing solution can be computed explicitly (e.g. by using the Moore-Penrose inverse of $N$).

We also note that $L_{\text{unique}}$ has a nice geometric description. In fact, minimizing $L_{\text{unique}}$ is essentially “centring” the convex body at the origin. To see this we first prove the following:

**Observation 1.** Let $H_h(x) = n_h \cdot x + d_h$ be a hyperplane and let $x_0$ be any point. Then the distance between $x_0$ and the hyperplane $H_h$ is given by $|H_h(x_0)|/\|n_h\|$.

**Proof.** Let $X$ be any point on the hyperplane $h$, so $H_h(X) = 0$. Then the distance of $x_0$ to $H_h$ is the norm of the projection of $X - x_0$ onto the norm of $H_h$, $n_h$. Using standard formulas we get, as desired:

$$||\text{Proj}_{n_h}(X - x_0)|| = \left|\frac{(x_0 - X) \cdot n_h}{\|n_h\|^2} - n_h\right|$$

$$= \left|\frac{n_h \cdot x_0 - n_h \cdot X}{\|n_h\|^2}\right|$$

$$= \left|\frac{n_h \cdot x_0 + d_h}{\|n_h\|}\right|$$

**□**

Note that if we assume that $n_h$ are all normalized, then $|H_h(x_0)|$ gives us the distance to the hyperplane $h$ for each hyperplane $h$. Therefore, given a convex polytope defined by collection of hyperplanes $h = \{(n_h, d_h)\}$, the $\sum_h |H_h(x)|^2$ is the sum of squares of distance of $x$ to the sides of the polytope. Also note that if we translate $h$ by $x_0$, then we get the new set of hyperplane $h' = \{(n_h, d_h + n_h \cdot x_0)\} = \{(n_h, H_h(x_0))\}$. Therefore,

$$L_{\text{unique}} = \sum_h ||H_h(0)||^2$$

can be interpreted as the square distance of the origin to the collection of hyperplanes $h$.  

13
Figure 21. **tSNE embedding** – We visualize the CvxNet (top) vs. the OccNet (bottom) tSNE latent space in 2D. We also visualize several examples with arrows taking them back to the CvxNet atlas and dashed arrows pointing them to the OccNet atlas. Notice how (1) nearby (distant) samples within the same class have a similar (dissimilar) geometric structure, and (2) the overlap between cabinets and speakers is meaningful as they both exhibit a cuboid geometry. Our interactive exploration of the t-SNE space revealed that our method produces meaningful embeddings, which are comparable to the ones from OccNet [44].