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 to real-time physics simulation in computer graphics, where it creates a unified representation of dynamic geometry for collision detection. 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. 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.

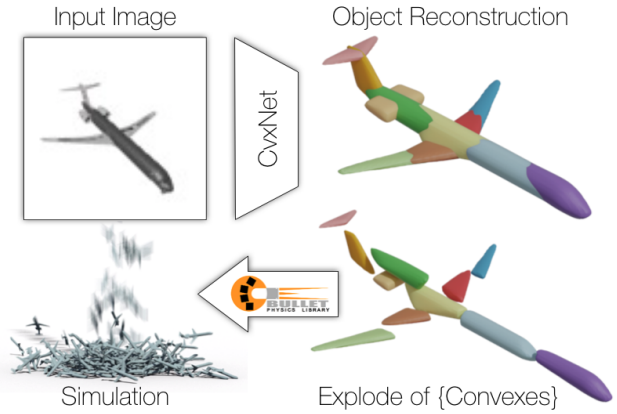


Figure 1. Our method reconstruct a 3D object from a input image as a collection of convex hulls. We visualize the explode of these convexes, which can then be readily used for physics simulation [17], as well as other downstream applications.

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, 56, 61, 68, 73], while hierarchical representations have attempted to reduce the memory footprint required for training [57, 63, 72], 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, 55, 70], 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 [67, 76], and their composition [59, 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 tessellation [50] or assembling geometry by smooth composition [2], while primitives enable highly efficient collision detection [65] and resolution [66]. 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 repre-

sentations are key to capturing 3D data whose topology is unknown [47, 46], while part-based models provide a natural decomposition of an object into its semantic components. This creates a representation useful to reason about extent, support, mass, contact, ... quantities that are key in order to describe the scene, and eventually design action plans [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 [67, 21, 43] we train our pipeline in a self-supervised 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 [67], ellipsoids [21], or sphere-meshes [66], 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 [42, 53, 60] models, due to their similarity to image based convolutions. Voxels have also been used successfully for generative models [74, 16, 24, 56, 61, 73]. However, the memory requirements of voxels makes them unsuitable for resolutions larger than 64^3. One can reduce the memory consumption significantly by using octrees that take advantage of the sparsity of voxels [57, 71, 72, 63]. This can extend the resolution to 512^3, 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 [41, 11, 27, 45]. Recently, meshes have also been considered as the output of a network [26, 32, 70]. 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. [52, 54] 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. [59] 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 [58], generalized cylinders [5], geons [4], and even Lego pieces [69]. In [67], 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 [49], they extended cuboids to *superquadrics*, showing that the extra flexibility will result in much lower error and qualitatively better looking results.

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 [43, 48, 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. [48] 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 geometry is missing from the final answer. In [21], they take a more geometric approach and restrict to level sets of restricted mixed Gaussians. Partly due to the restrictions of the functions they use, their representation struggles on shapes with angled parts, but they do recover the interpretability of [67] by considering the level set of each Gaussian component individually.

Convex decomposition In graphics, a common method to

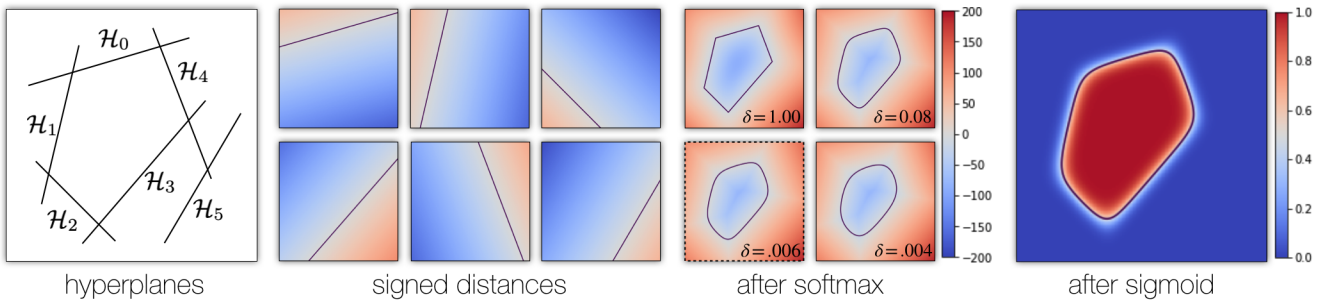


Figure 2. An example of how a collection of hyperplane parameters for an image specifies the indicator *function* of a convex object. The soft-max allows gradients to propagate through all hyperplanes and allows for the generation of *smooth* convex, while the sigmoid parameter controls the *slope* of the transition in the generated indicator. Note that our soft-max function is a LogSumExp.

representing shapes is to describe them as a collection of convex objects. Several methods for convex decomposition of meshes have been proposed [25, 51]. In machine learning, however, we only find some initial attempts to approach convex hull computation via neural networks [34]. Splitting the meshes into exactly convex parts generally produces too many pieces [13]. As such, it is more prudent to seek small number of approximately convex objects that generate the input shape [22, 36, 38, 40, 39]. Recently [65] also extended convex decomposition to the spatio-temporal domain, by considering moving geometry. Our method is most related to [67] and [21], in that we train an occupancy function for the input shapes. 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 $\mathcal{O} : \mathbb{R}^3 \to [0, 1]$, and with $\partial\mathcal{O} = \{\mathbf{x} \in \mathbb{R}^3 \mid \mathcal{O}(x) = 0.5\}$ we indicate the surface of the object. The indicator function is defined such that $\{\mathbf{x} \in \mathbb{R}^3 \mid \mathcal{O}(x) = 0\}$ defines the outside of the object and $\{\mathbf{x} \in \mathbb{R}^3 \mid \mathcal{O}(x) = 1\}$ the inside. Given an input (*e.g.* an image, point cloud, or voxel grid) an encoder estimates the parameters $\{\boldsymbol{\beta}_k\}$ of our template representation $\hat{\mathcal{O}}(\cdot)$ with K primitives (indexed by k). We then evaluate the template at random sample points $\mathbf{x}$, and our training loss ensures $\hat{\mathcal{O}}(\mathbf{x}) \approx \mathcal{O}(\mathbf{x})$. In the discussion below, without loss of generality, we use 2D illustrative examples where $\mathcal{O} : \mathbb{R}^2 \to [0, 1]$. 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 3.3). We then describe the losses used to train our networks (Section 3.4) and mention a few implementation details (Section 3.5).

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 $\mathbf{x} \in \mathbb{R}^3$. We define $\mathcal{H}_h(\mathbf{x}) = \mathbf{n}_h \cdot \mathbf{x} + d_h$ as the signed distance of the point $\mathbf{x}$ from the h-th plane with normal $\mathbf{n}_h$ 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 intersection (max operator) 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, $\Phi(x)$:

$$\Phi(\mathbf{x}) = \text{LogSumExp}\{\delta\mathcal{H}_h(\mathbf{x})\}, \tag{1}$$

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

$$\mathcal{C}(\mathbf{x}|\boldsymbol{\beta}) = \text{Sigmoid}(-\sigma\Phi(\mathbf{x})), \tag{2}$$

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

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

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 possible

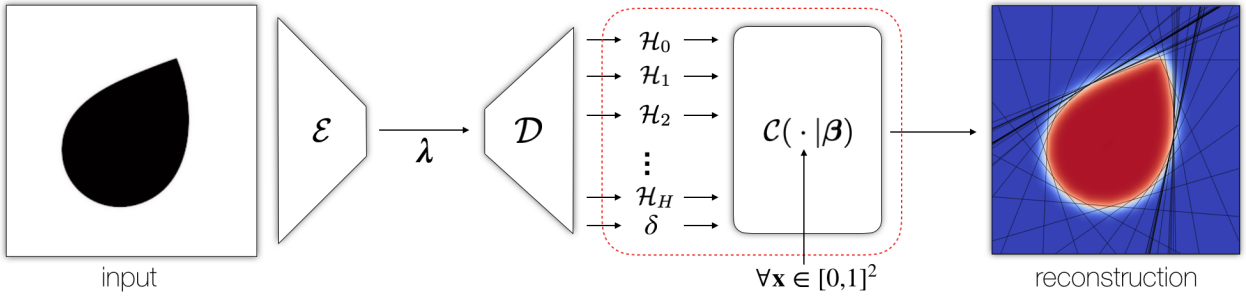


Figure 3. **Convex auto-encoder** – The encoder $\mathcal{E}$ creates a low dimensional latent vector representation λ, decoded into a collection of hyperplanes by the decoder $\mathcal{D}$. The training loss involves reconstructing the value of the input image at random pixels $\mathbf{x}$.

to discover some form of correlation between their parameters. Towards this goal, we employ the bottleneck auto-encoder architecture illustrated in Figure 3. Given an input, the encoder $\mathcal{E}$ derives a latent representation λ from the input. Then, a decoder $\mathcal{D}$ derives the collection of hyperplane parameters. While in theory permuting the H hyperplanes generates the same convex, the decoder $\mathcal{D}$ 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 [67] and SIF [21]; we verify this claim in Figure 5.

3.3. Multi convex decomposition – Figure 6

Having a learnable pipeline for a single convex object, we can now generalize the expressivity of our model by representing generic *non-convex* objects as *compositions* of convex elements [65]. To achieve this task an encoder $\mathcal{E}$ outputs a low-dimensional latent representation of all K convexes $\boldsymbol{\lambda}$ that $\mathcal{D}$ decodes into a *collection* of K parameter tuples. Each tuple (indexed by k) is comprised of a shape code $\boldsymbol{\beta}_k$, and corresponding transformation $\mathbf{T}_k(\mathbf{x}) = \mathbf{x} + \mathbf{c}_k$ that transforms the point from world coordinates to local coordinates. $\mathbf{c}_k$ is the predicted translation vector (Figure 6).

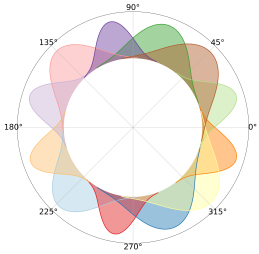


Figure 4. **Correlation** – While the description of a convex, $\{(\mathbf{n}_h, \mathbf{d}_h)\}$, is permutation invariant we employ an encoder/decoder that implicitly establishes an ordering. Our visualization reveals how a particular hyperplane typically represents a particular subset of orientations.

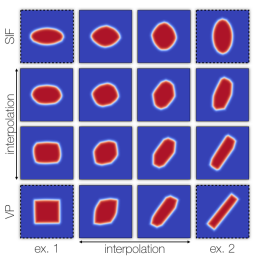


Figure 5. **Interpolation** – We compute latent code of shapes in the corners using CvxNet. We then linearly interpolate latent codes to synthesize shapes in-between. Our primitives generalize the shape space of VP [67] (boxes) and SIF [21] (ellipsoids) so we can interpolate between them smoothly.

3.4. Training losses

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

$$\mathcal{L}_{\text{approx}}(\omega) = \mathbb{E}_{\mathbf{x}\sim\mathbb{R}^3} \|\hat{\mathcal{O}}(\mathbf{x}) - \mathcal{O}(\mathbf{x})\|^2, \tag{3}$$

where $\hat{\mathcal{O}}(\mathbf{x}) = \max_k\{\mathcal{C}_k(\mathbf{x})\}$, and $\mathcal{C}_k(\mathbf{x}) = \mathcal{C}(\mathbf{T}_k(\mathbf{x})|\boldsymbol{\beta}_k)$. The application of the $\max$ operator produces a perfect union of indicator functions. 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 Appendix A for more details. We couple the approximation loss with a small set of auxiliary losses that enforce the desired properties of our decomposition.

Decomposition loss (auxiliary). We seek a parsimonious decomposition of an object alike Tulsiani et al. [67]. Hence, overlap between elements should be discouraged:

$$\mathcal{L}_{\text{decomp}}(\omega) = \mathbb{E}_{\mathbf{x}\sim\mathbb{R}^3} \|\text{relu}(\underset{k}{\text{sum}}\{\mathcal{C}_k(\mathbf{x})\} - \tau)\|^2, \tag{4}$$

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 $\mathbf{T}$ accordingly – see Figure 7(left). To remove such a null-space, we simply regularize the magnitudes of the offsets for each of the K elements:

$$\mathcal{L}_{\text{unique}}(\omega) = \tfrac{1}{H} \sum_h \|d_h\|^2 \tag{5}$$

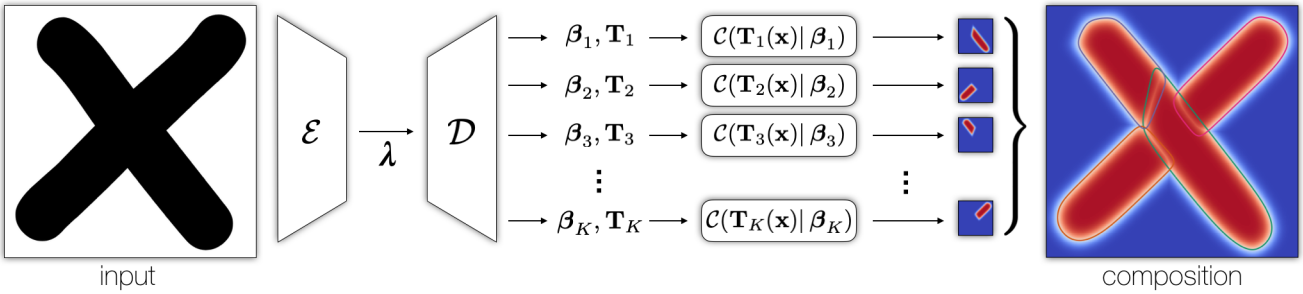


Figure 6. **Multi-convex auto-encoder** – Our network approximates input geometry as a *composition* of convex elements. Note that this network *does not* prescribe how the final image is generated, but merely output the shape $\{\boldsymbol{\beta}_k\}$ and pose $\{\mathbf{T}_k\}$ parameters of the abstraction. Note that this is an illustration where the parameters $\{\boldsymbol{\beta}_k\}, \{\boldsymbol{T}_k\}$ have been directly optimized via SGD with a preset δ.

In the supplementary material, we prove that minimizing $\mathcal{L}_{\text{unique}}$ 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 3.5, 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. $\nexists x \mid \mathcal{C}(x) > 0.5$), 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 a certain amount of information (i.e. samples):

$$\mathcal{L}_{\text{guide}}(\omega) = \frac{1}{K} \sum_{k} \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{N}_k^N} \|\mathcal{C}_k(\mathbf{x}) - \mathcal{O}(\mathbf{x})\|^2, \qquad (6)$$

where $\mathcal{N}_k^N$ is the subset of N samples from the set $\mathbf{x} \sim \{\mathcal{O}\}$ with the smallest distance value $\Phi_k(\mathbf{x})$ from $\mathcal{C}_k$. In other words, each convex is responsible for representing *at least* the N closest interior samples.

Localization loss (auxiliary). When a convex is far from interior points, the loss in (6) suffers from vanishing gradients due to the sigmoid function. We overcome this problem by adding a loss with respect to $\mathbf{c}_k$, the translation vector of the k-th convex:

$$\mathcal{L}_{\text{loc}}(\omega) = \frac{1}{K} \sum_{\mathbf{x} \in \mathcal{N}_k^1} \|\mathbf{c}_k - x\|^2 \qquad (7)$$

Observations. Note that we supervise the indicator function $\mathcal{C}$ rather than Φ, as the latter *does not* represent the signed distance function of a convex (e.g. $\|\nabla\Phi(x)\| \neq 1$). Also note how the loss in (4) is reminiscent of SIF [21, Eq.1], where the overall surface is modeled as a sum of *meta-ball* implicit functions [7] – which the authors call "structuring". The core difference lies in the fact that SIF [21] models the surface of the object $\partial\mathcal{O}$ as an iso-level of the function *post* structuring – therefore, in most cases, the iso-surface of the individual primitives do not approximate the target surface, resulting in a slight loss of interpretability in the generated representation.

3.5. Implementation details

To increase training speed, we sample a set of points on the ground-truth shape offline, precompute the ground truth quantities, and then randomly sub-sample from this set during our training loop. For volumetric samples, we use the samples from OccNet [43], while for surface samples we employ the "near-surface" sampling described in SIF [21]. Following SIF [21], we also tune down $\mathcal{L}_{\text{approx}}$ of "near-surface" samples by 0.1. We draw $100k$ random samples from the bounding box of $\mathcal{O}$ and $100k$ samples from each

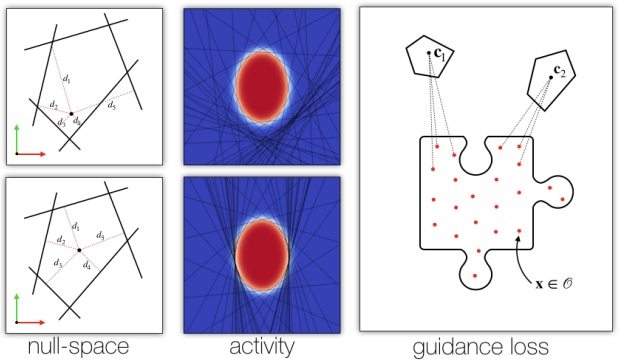


Figure 7. **Auxiliary losses –** Our $\mathcal{L}_{\text{unique}}$ loss (left) prevents the existence of a null-space in the specification of convexes, and (middle) ensures inactive hyperplanes can be easily activated during training. (right) Our $\mathcal{L}_{\text{guide}}$ move convexes towards the representation of samples drawn from within the object $\mathbf{x} \in \mathcal{O}$.

of $\partial\mathcal{O}$ to construct the points samples and labels. We use a sub-sample set (at training time) with 1024 points for both sample sources. Although Mescheder et al. [43] claims that using uniform volumetric samples are more effective than surface samples, we find that balancing these two strategies yields the best performance – this can be attributed to the complementary effect of the losses in (3) and (4).

Architecture details. In all our experiments, we use the same architecture while varying the number of convexes and hyperplanes. For the {Depth}-to-3D task, we use 50 convexes each with 50 hyperplanes. For the RGB-to-3D task, we use 50 convexes each with 25 hyperplanes. Similar to OccNet [43], we use ResNet18 as the encoder $\mathcal{E}$ for both the {Depth}-to-3D and the RGB-to-3D experiments. A fully connected layer then generates the latent code $\boldsymbol{\lambda} \in \mathbb{R}^{256}$ that is provided as input to the decoder $\mathcal{D}$. For the decoder $\mathcal{D}$ 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 $1e-4$, and $\beta_1 = .9$ and $\beta_2 = .999$. As determined by grid-search on the validation set, we set the weight for our losses $\{\mathcal{L}_{\text{approx}} : 1.0, \mathcal{L}_{\text{decomp}} : 0.1, \mathcal{L}_{\text{unique}} : 0.001, \mathcal{L}_{\text{guide}} : 0.01, \mathcal{L}_{\text{loc}} : 1.0\}$ and $\sigma = 75$.

4. Experiments

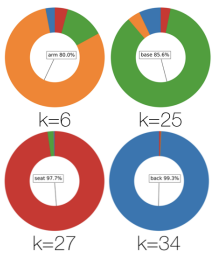
We use the ShapeNet [12] dataset in our experiments. We use the same voxelization, renderings, and data split as in Choy et. al. [16]. Moreover, we use the same multi-view depth renderings as [21] for our {Depth}-to-3D experiments, where we render each example from cameras placed on the vertices of a dodecahedron. Note that this problem is a harder problem than 3D auto-encoding with point cloud input as proposed by OccNet [43] 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 [43] – this also ensures a fair comparison to this method. For the quantitative evaluation of semantic decomposition, we use labels from PartNet [44] 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 [67] 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 [67], and in contrast to OccNet [43], this methods provides an *interpretable* encoding of geometry. Finally, from the class of techniques that *directly* learn *non-interpretable* representations of implicit functions, we select OccNet [43], P2M [70], and AtlasNet [26]; in contrast to the previous methods, these solutions do not provide any form of shape decomposition. As OccNet [43] 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 $\hat{\mathcal{O}}$ and $\partial\hat{\mathcal{O}}$ we respectively indicate the indicator and 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 32^3 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* $\mathbb{E}_{\mathbf{o}\sim\partial\mathcal{O}}[\min_{\hat{\mathbf{o}}\in\partial\hat{\mathcal{O}}} \|\mathbf{o}-\hat{\mathbf{o}}\|]$ and *completeness* $\mathbb{E}_{\hat{\mathbf{o}}\sim\partial\hat{\mathcal{O}}}[\min_{\mathbf{o}\in\partial\mathcal{O}} \|\hat{\mathbf{o}}-\mathbf{o}\|]$ [18]. ③ Following the arguments presented in [64], we also employ F-score to quantitatively assess performance. It can be understood as "the percentage of correctly reconstructed surface".

4.1. Abstraction – Figure 8, 9, 10

As our convex decomposition is learnt on a shape collection, the convex elements 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. We do so by verifying whether the k-th component is consistently mapped to the same PartNet part label; see Figure 8 (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 8 (right). This reveals how our latent representation captures the semantic structure in the dataset. We also evaluate our shape



Part	Accuracy Cvx	Accuracy BAE	Accuracy BAE*
back	**91.50%**	86.36%	91.81%
arm	38.94%	**65.75%**	71.32%
base	71.95%	**88.46%**	91.75%
seat	**90.63%**	73.66%	92.91%

Figure 8. (left) The distribution of partnet labels within each convex ID (4 out of 50). (right) The binary classification accuracy for each semantic part when using the convex ID to label each point. Cvx represents CvxNet. BAE [14] is a baseline for unsupervised part segmentation. BAE* is the supervised version of BAE.

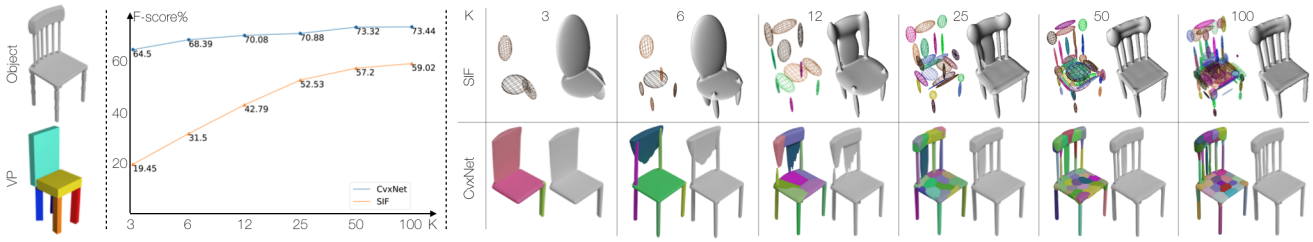


Figure 9. **Analysis of accuracy vs. # primitives** – (left) The ground truth object to be reconstructed and the single shape-abstraction generated by VP [67]. (middle) Quantitative evaluation (ShapeNet/Multi) of abstraction performance with an increase number of primitives – the closer the curve is to the top-left, the better. (right) A qualitative visualization of the primitives and corresponding reconstructions.

abstraction capabilities by varying the number of components and evaluate 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 *part 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 [43]) when evaluated

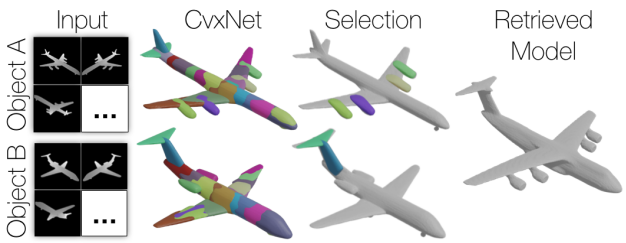


Figure 10. **Part based retrieval** – Two inputs (left) are first encoded into our CvxNet representation (middle-left), from which a user can select a subset of parts (middle-right). We then use the concatenated latent code as an (incomplete) geometric lookup function, and retrieve the closest decomposition in the training database (right).

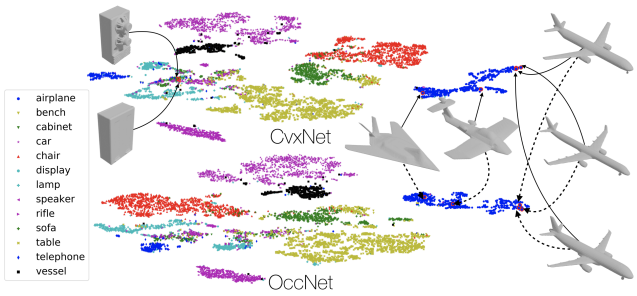


Figure 11. **Shape embedding** – We visualize (top) CvxNet vs. (bottom) OccNet latent space via a 2D tSNE embedding. We also visualize several examples with arrows taking them back to the CvxNet atlas and dashed arrows pointing them to the OccNet atlas.

in terms of F-score, and tested on multi-view depth input. Note that SIF [21] first trains for the template parameters on ({Depth}-to-3D) with a reconstruction loss, and then trains the RGB-to-3D image encoder with a parameter regression loss; conversely, our method trains both encoder and decoder of the RGB-to-3D task from *scratch*.

4.3. Latent space analysis – Figure 11

We investigate the latent representation learnt by our network by computing a t-SNE embedding. 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 comparable to OccNet [43]; we illustrate this qualitatively in Figure 11.

4.4. 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 chosen size of the latent space $|\boldsymbol{\lambda}|$. We also investigate the effect of varying the number of convexes K and number of hyperplanes H

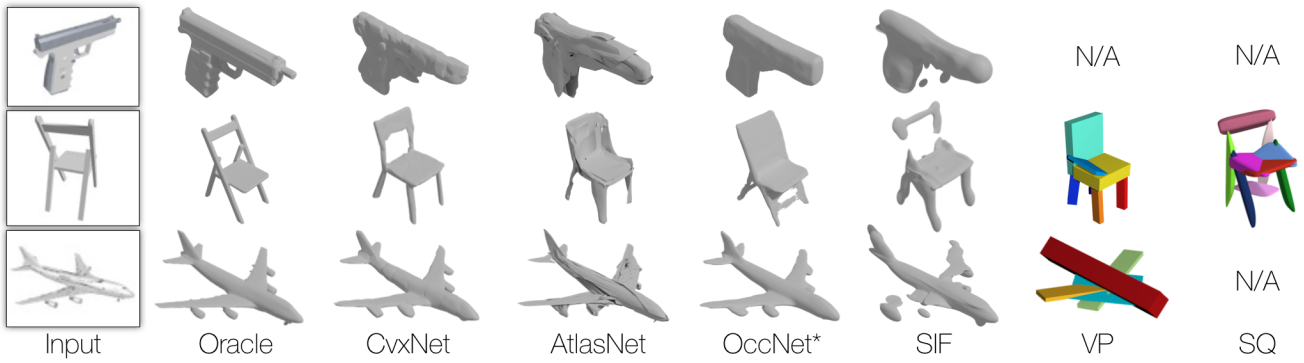


Figure 12. **ShapeNet/Multi** – Qualitative comparisons to SIF [21], AtlasNet [26], OccNet [43], VP [67] and SQ [49]; on RGB Input, while VP uses voxelized, and SQ uses a point-cloud input. (*Note that the OccNet [43] results are post-processed with smoothing).

Category	IoU			Chamfer-L_1			F-Score		
	OccNet	SIF	Ours	OccNet	SIF	Ours	OccNet	SIF	Ours
airplane	0.728	0.662	**0.739**	0.031	0.044	**0.025**	79.52	71.40	**84.68**
bench	**0.655**	0.533	0.631	**0.041**	0.082	0.043	71.98	58.35	**77.68**
cabinet	**0.848**	0.783	0.830	0.138	0.110	**0.048**	71.31	59.26	**76.09**
car	**0.830**	0.772	0.826	0.071	0.108	**0.031**	69.64	56.58	**77.75**
chair	**0.696**	0.572	0.681	0.124	0.154	**0.115**	63.14	42.37	**65.39**
display	**0.763**	0.693	0.762	0.087	0.097	**0.065**	63.76	56.26	**71.41**
lamp	**0.538**	0.417	0.494	0.678	**0.342**	0.352	**51.60**	35.01	51.37
speaker	**0.806**	0.742	0.784	0.440	0.199	**0.112**	58.09	47.39	**60.24**
rifle	0.666	0.604	**0.684**	0.033	0.042	**0.023**	78.52	70.01	**83.63**
sofa	**0.836**	0.760	0.828	0.052	0.080	**0.036**	69.66	55.22	**75.44**
table	**0.699**	0.572	0.660	0.152	0.157	**0.121**	68.80	55.66	**71.73**
phone	**0.885**	0.831	0.869	0.022	0.039	**0.018**	85.60	81.82	**89.28**
vessel	**0.719**	0.643	0.708	0.070	0.078	**0.052**	66.48	54.15	**70.77**
mean	**0.744**	0.660	0.731	0.149	0.118	**0.080**	69.08	59.02	**73.49**

{Depth}-to-3D

Category	IoU					Chamfer-L_1					F-Score			
	P2M	AtlasNet	OccNet	SIF	Ours	P2M	AtlasNet	OccNet	SIF	Ours	AtlasNet	OccNet	SIF	Ours
airplane	0.420	-	0.571	0.530	**0.598**	0.187	0.104	0.147	0.167	**0.093**	67.24	62.87	52.81	**68.16**
bench	0.323	-	**0.485**	0.333	0.461	0.201	0.138	0.155	0.261	**0.133**	54.50	**56.91**	37.31	54.64
cabinet	0.664	-	**0.733**	0.648	0.709	0.196	0.175	0.167	0.233	**0.160**	46.43	**61.79**	31.68	46.09
car	0.552	-	**0.737**	0.657	0.675	0.180	0.141	0.159	0.161	**0.103**	51.51	**56.91**	37.66	47.33
chair	0.396	-	**0.501**	0.389	0.491	0.265	**0.209**	0.228	0.380	0.337	38.89	**42.41**	26.90	38.49
display	0.490	-	0.471	0.491	**0.576**	0.239	**0.198**	0.278	0.401	0.223	**42.79**	38.96	27.22	40.69
lamp	0.323	-	**0.371**	0.260	0.311	0.308	**0.305**	0.479	1.096	0.795	33.04	**38.35**	20.59	31.41
speaker	0.599	-	**0.647**	0.577	0.620	0.285	**0.245**	0.300	0.554	0.462	35.75	**42.48**	22.42	29.45
rifle	0.402	-	0.474	0.463	**0.515**	0.164	0.115	0.141	0.193	**0.106**	**64.22**	56.52	53.20	63.74
sofa	0.613	-	**0.680**	0.606	0.677	0.212	0.177	0.194	0.272	**0.164**	43.46	**48.62**	30.94	42.11
table	0.395	-	**0.506**	0.372	0.473	0.218	0.190	**0.189**	0.454	0.358	44.93	**58.49**	30.78	48.10
phone	0.661	-	**0.720**	0.658	0.719	0.149	0.128	0.140	0.159	**0.083**	58.85	**66.09**	45.61	59.64
vessel	0.397	-	0.530	0.502	**0.552**	0.212	**0.151**	0.218	0.208	0.173	**49.87**	42.37	36.04	45.88
mean	0.480	-	**0.571**	0.499	0.567	0.216	**0.175**	0.215	0.349	0.245	48.57	**51.75**	34.86	47.36

RGB-to-3D

Table 1. **Reconstruction performance on ShapeNet/Multi** – We evaluate our method against P2M [70], AtlasNet [26], OccNet [43] 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 [67], as it only produces a very rough shape decomposition.

in terms of reconstruction accuracy and inference/training time. Moreover, we quantitatively demonstrate that using signed distance as supervision for $\mathcal{L}_{\text{approx}}$ produces significantly worse results and at the cost of slightly worse performance we can collapse $\mathcal{L}_{\text{guide}}$ and $\mathcal{L}_{\text{loc}}$ 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 [43] in Table 1). In the future we would like to generalize the model to be able to predict a variable number of parts [67], understand symmetries and modeling hierarchies [75], and include the modeling of rotations [67]. Leveraging the invariance of hyperplane ordering, it would be interesting to investigate the effect of permutation-invariant encoders [62], or remove encoders altogether in favor of *auto-decoder* architectures [48].

A. Union of Smooth Indicator Functions

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

$$\mathcal{C}_k(\mathbf{x}) = \text{Sigmoid}_\sigma(-\Phi^k_\delta(\mathbf{x})), \tag{8}$$

where $\Phi^k_\delta(\mathbf{x})$ 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:

$$\begin{aligned}
\mathcal{U}\{\mathcal{C}_k(\mathbf{x})\} &= \text{Sigmoid}_\sigma(-\min_k\{\Phi^k_\delta(\mathbf{x})\}) && (9)\\
&= \text{Sigmoid}_\sigma(\max_k\{-\Phi^k_\delta(\mathbf{x})\}) && (10)\\
&= \max_k\{\text{Sigmoid}_\sigma(\Phi^k_\delta(\mathbf{x}))\} && (11)\\
&= \max_k\{\mathcal{C}_k(\mathbf{x})\}. && (12)
\end{aligned}$$

Note that the $\max$ operator is *commutative* with respect to monotonically increasing functions allowing us to extract the $\max$ operator from the $\text{Sigmoid}_\sigma(\cdot)$ function in (11).

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International Conference on Machine Learning*, pages 40–49, 2018. 1

[2] Baptiste Angles, Marco Tarini, Loic Barthe, Brian Wyvill, and Andrea Tagliasacchi. Sketch-based implicit blending. *ACM Transaction on Graphics (Proc. SIGGRAPH Asia)*, 2017. 1

[3] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, volume 36, pages 301–329. Wiley Online Library, 2017. 2

[4] Irving Biederman. Recognition-by-components: a theory of human image understanding. *Psychological review*, 1987. 1, 2

[5] Thomas Binford. Visual perception by computer. In *IEEE Conference of Systems and Control*, 1971. 2

[6] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3D faces. In *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 1999. 1

[7] James F Blinn. A generalization of algebraic surface drawing. *ACM Trans. on Graphics (TOG)*, 1(3):235–256, 1982. 5

[8] Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J Black. Keep it smpl: Automatic estimation of 3d human pose and shape from a single image. In *Proc. of the European Conf. on Comp. Vision*, 2016. 1

[9] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. AK Peters/CRC Press, 2010. 1

[10] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016. 1

[11] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 2

[12] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 6

[13] Bernard M Chazelle. Convex decompositions of polyhedra. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 70–79. ACM, 1981. 3

[14] Zhiqin Chen, Kangxue Yin, Matthew Fisher, Siddhartha Chaudhuri, and Hao Zhang. Bae-net: Branched autoencoder for shape co-segmentation. *Proceedings of International Conference on Computer Vision (ICCV)*, 2019. 6

[15] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *Proc. of Comp. Vision and Pattern Recognition (CVPR)*, 2019. 2

[16] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proc. of the European Conf. on Comp. Vision*. Springer, 2016. 2, 6

[17] Erwin Coumans and Yunfei Bai. PyBullet, a python module for physics simulation for games, robotics and machine learning. pybullet.org, 2016–2019. 1, 12

[18] Siyan Dong, Matthias Niessner, Andrea Tagliasacchi, and Kevin Kai Xu. Multi-robot collaborative dense scene reconstruction. *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 2019. 6

[19] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017. 1, 2

[20] Matheus Gadelha, Subhransu Maji, and Rui Wang. 3d shape induction from 2d views of multiple objects. In *International Conference on 3D Vision (3DV)*, 2017. 1

[21] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. *arXiv preprint arXiv:1904.06447*, 2019. 1, 2, 3, 4, 5, 6, 7, 8

[22] Mukulika Ghosh, Nancy M Amato, Yanyan Lu, and Jyh-Ming Lien. Fast approximate convex decomposition using relative concavity. *Computer-Aided Design*, 45(2):494–504, 2013. 3

[23] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 1988. 2

[24] Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *Proc. of the European Conf. on Comp. Vision*, pages 484–499. Springer, 2016. 2

[25] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Info. Pro. Lett.*, 1:132–133, 1972. 3

[26] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mache approach to learning 3d surface generation. In *Proc. of Comp. Vision and Pattern Recognition (CVPR)*, 2018. 1, 2, 6, 8

[27] Kan Guo, Dongqing Zou, and Xiaowu Chen. 3d mesh labeling via deep convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 35(1):3, 2015. 2

[28] Eric Heiden, David Millard, and Gaurav Sukhatme. Real2sim transfer using differentiable physics. Workshop on Closing the Reality Gap in Sim2real Transfer for Robotic Manipulation, 2019. 2

[29] Eric Heiden, David Millard, Hejia Zhang, and Gaurav S Sukhatme. Interactive differentiable simulation. *arXiv preprint arXiv:1905.10706*, 2019. 2

[30] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 71–78. ACM, 1992. 2

[31] Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Learning category-specific mesh reconstruc-

tion from image collections. In *Proc. of the European Conf. on Comp. Vision*, 2018. 1

[32] Chen Kong, Chen-Hsuan Lin, and Simon Lucey. Using locally corresponding cad models for dense 3d reconstructions from a single image. In *Proc. of Comp. Vision and Pattern Recognition (CVPR)*, pages 4857–4865, 2017. 2

[33] David H Laidlaw, W Benjamin Trumbore, and John F Hughes. Constructive solid geometry for polyhedral objects. In *ACM Trans. on Graphics (Proc. of SIGGRAPH)*, 1986. 2

[34] Yee Leung, Jiang-She Zhang, and Zong-Ben Xu. Neural networks for convex hull computation. *IEEE Transactions on Neural Networks*, 8(3):601–611, 1997. 3

[35] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proc. of Comp. Vision and Pattern Recognition (CVPR)*, 2018. 1

[36] Jyh-Ming Lien and Nancy M Amato. Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 121–131. ACM, 2007. 3

[37] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 2

[38] Guilin Liu, Zhonghua Xi, and Jyh-Ming Lien. Nearly convex segmentation of polyhedra through convex ridge separation. *Computer-Aided Design*, 78:137–146, 2016. 3

[39] Khaled Mamou and Faouzi Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *2009 16th IEEE international conference on image processing (ICIP)*, pages 3501–3504. IEEE, 2009. 3

[40] Khaled Mamou, E Lengyel, and Ed AK Peters. Volumetric hierarchical approximate convex decomposition. *Game Engine Gems 3*, pages 141–158, 2016. 3

[41] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015. 2

[42] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015. 2

[43] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. *arXiv preprint arXiv:1812.03828*, 2018. 2, 5, 6, 7, 8

[44] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 909–918, 2019. 6

[45] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. of Comp. Vision and Pattern Recognition (CVPR)*, pages 5115–5124, 2017. 2

[46] Richard A Newcombe, Dieter Fox, and Steven M Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015. 2

[47] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Proc. ISMAR*. IEEE, 2011. 2

[48] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *arXiv preprint arXiv:1901.05103*, 2019. 2, 8

[49] Despoina Paschalidou, Ali Osman Ulusoy, and Andreas Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2, 8

[50] Jason Patnode. *Character Modeling with Maya and ZBrush: Professional polygonal modeling techniques*. Focal Press, 2012. 1

[51] Franco P Preparata and Se June Hong. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20(2):87–93, 1977. 3

[52] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 2

[53] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016. 2

[54] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. 2

[55] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J Black. Generating 3d faces using convolutional mesh autoencoders. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 1

[56] Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. In *Advances in Neural Information Processing Systems*, 2016. 1, 2

[57] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017. 1, 2

[58] Lawrence G Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963. 2

[59] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *Proc. of Comp. Vision and Pattern Recognition (CVPR)*, 2018. 1, 2

[60] Shuran Song and Jianxiong Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *Proceed-*

ings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 808–816, 2016. 2

[61] David Stutz and Andreas Geiger. Learning 3d shape completion from laser scan data with weak supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1955–1964, 2018. 1, 2

[62] Weiwei Sun, Wei Jiang, Eduard Trulls, Andrea Tagliasacchi, and Kwang Moo Yi. Attentive context normalization for robust permutation-equivariant learning. *arXiv preprint arXiv:1907.02545*, 2019. 8

[63] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2088–2096, 2017. 1, 2

[64] Maxim Tatarchenko, Stephan R Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3d reconstruction networks learn? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3405–3414, 2019. 6

[65] Daniel Thul, Sohyeon Jeong, Marc Pollefeys, et al. Approximate convex decomposition and transfer for animated meshes. In *SIGGRAPH Asia 2018 Technical Papers*, page 226. ACM, 2018. 1, 3, 4

[66] Anastasia Tkach, Mark Pauly, and Andrea Tagliasacchi. Sphere-meshes for real-time hand modeling and tracking. *ACM Transaction on Graphics (Proc. SIGGRAPH Asia)*, 2016. 1, 2

[67] Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 1, 2, 3, 4, 6, 7, 8

[68] Ali Osman Ulusoy, Andreas Geiger, and Michael J Black. Towards probabilistic volumetric reconstruction using ray potentials. In *International Conference on 3D Vision (3DV)*, 2015. 1

[69] Anton van den Hengel, Chris Russell, Anthony Dick, John Bastian, Daniel Pooley, Lachlan Fleming, and Lourdes Agapito. Part-based modelling of compound scenes from images. In *Proc. of Comp. Vision and Pattern Recognition (CVPR)*, pages 878–886, 2015. 2

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

[71] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72, 2017. 2

[72] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. Adaptive o-cnn: a patch-based deep representation of 3d shapes. In *SIGGRAPH Asia 2018 Technical Papers*, page 217. ACM, 2018. 1, 2

[73] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems*, pages 82–90, 2016. 1, 2

[74] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proc. of Comp. Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. 2

[75] Fenggen Yu, Kun Liu, Yan Zhang, Chenyang Zhu, and Kai Xu. Partnet: A recursive part decomposition network for fine-grained and hierarchical shape segmentation. *Proc. of Comp. Vision and Pattern Recognition (CVPR)*, 2019. 8

[76] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017. 1

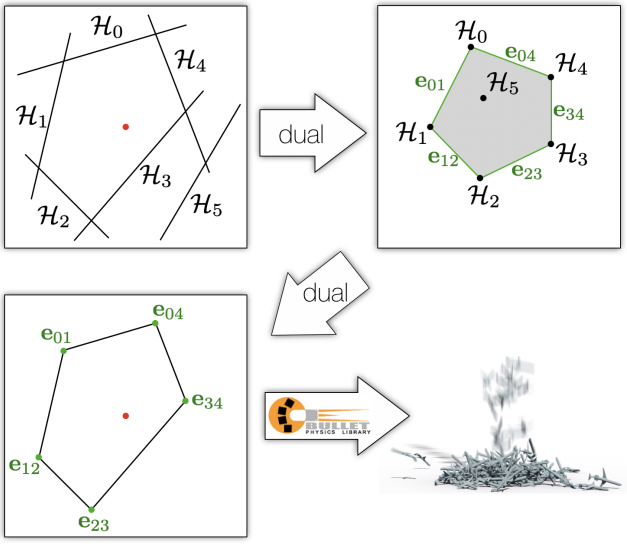


Figure 13. **CvxNet for physics simulation** – A zoom-in of the simulation pipeline used to generate the example in Figure 1. The animated results can be found in the **supplementary video**. For each convex, the hyperplane equations are converted via a "geometric dual" to a collection of points; the convexes to be used in simulation are then computed as the convex-hull of these points. Unused hyperplanes, e.g. $\mathcal{H}_5$, become points inside the convex-hull. Then we take the inverse dual to get the convex-hull in the original space. This convex-hull can be used by physics engines to simulate animations. The simulation is computed and rendered by Houdini, via its rigid-body world solver based on Bullet [17].

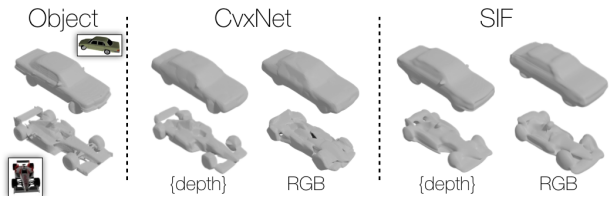


Figure 14. **Depth vs. Color** – A qualitative example illustrating the degradation in performance as we move from {depth} to a "weaker" RGB input. In the top row we illustrate a model where the frequency of the surface is low. In this case, both {depth} input and RGB input can approximate the shape 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.

A. 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 training strategy to merge these 2 losses into:

$$\mathcal{L}_{\text{merged}} = \frac{1}{K}\sum_{k}\frac{1}{N}\sum_{\mathbf{x}\in\mathcal{N}_k}\|\text{ReLU}(\Phi_k(\mathbf{x}))\|^2, \qquad (13)$$

The idea is that while $\Phi_k(\mathbf{x})$ is approximate, it can still be used to get weak gradients. Essentially, this means that each

Losses	IoU	Chamfer-L_1	F-Score
Original	**0.731**	**0.080**	**73.49%**
Merged	0.720	0.087	71.62%

Figure 15. **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 A. At the cost of slightly worse performance, we can simplify the training objective.

Class	# exemplars	{Depth}-to-3D		RGB-to-3D	
		F% Single	F% Multi	F% Single	F% Multi
table	5958	70.97	**71.71**	**50.29**	48.10
car	5248	77.11	**77.75**	**51.34**	47.33
chair	4746	62.21	**65.39**	38.12	**38.49**
plane	2832	83.68	**84.68**	**75.19**	68.16
sofa	2222	67.89	**75.44**	**43.36**	42.11
rifle	1661	82.73	**83.63**	**69.62**	63.74
lamp	1624	46.46	**51.37**	**32.49**	31.41
vessel	1359	65.71	**70.77**	**48.44**	45.88
bench	1272	68.20	**77.68**	**59.27**	54.64
speaker	1134	50.37	**60.24**	28.07	**28.45**
cabinet	1101	66.75	**76.09**	45.73	**46.09**
display	767	61.66	**71.41**	**40.31**	38.96
phone	737	84.93	**89.28**	63.58	**66.09**
mean	2359	68.36	**73.50**	**49.68**	47.65

Figure 16. **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.

K \ H	12	25	50
12	0.709 / 0.139 / **44**	0.712 / 0.127 / 59	0.714 / 0.124 / 87
25	0.717 / 0.100 / 60	0.720 / 0.099 / 92	0.721 / 0.096 / 153
50	0.724 / 0.088 / 92	0.730 / 0.083 / 156	**0.731** / **0.080** / 280

Figure 17. **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.

Metric \ Loss	All	$-\mathcal{L}_{\text{decomp}}$	$-\mathcal{L}_{\text{unique}}$	$-\mathcal{L}_{\text{guide}}$	$-\mathcal{L}_{\text{loc}}$
Vol-IoU	**0.567**	0.558	0.545	0.551	0.558
Chamfer	**0.245**	0.308	0.313	0.335	0.618
F%	**47.36**	45.29	44.03	45.88	46.01

Figure 18. **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.

Supervision	IoU	Chamfer-L_1	F-Score
Indicator	**0.747**	**0.035**	**83.22%**
SDF	0.650	0.045	73.08%

Figure 19. **Ablation SDF Training vs. Indicator Training** – We study the difference between learning signed distance functions(i.e., replacing $\mathcal{O}(\mathbf{x})$ 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.

$\lvert\boldsymbol{\lambda}\rvert$	F-Score
32	72.24%
64	73.31%
128	73.14%
256	**73.49%**

Figure 20. **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.

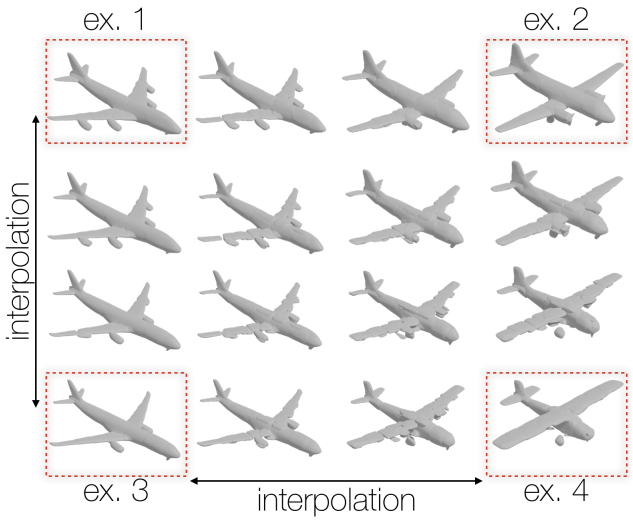


Figure 21. **Shape interpolation** – Analogously to Figure 5, we encode the four models in the corners, and generate the intermediate results via linear interpolation. These results can be better appreciated in the **supplementary video** where we densely interpolate between the corner examples.

convex needs to "explain" the N closest samples of $\mathcal{O}$.

B. Proof of auxiliary null-space loss

Consider a hyperplane $\mathcal{H}_h(\mathbf{x}) = \mathbf{n}_h \cdot \mathbf{x} + d_h$, parametrized by a pair $(\mathbf{n}_h, d_h)$. Translating this hyperplane by $\mathbf{x}_0$ gives us the hyperplane $(\mathbf{n}_h, d_h + \mathbf{n}_h \cdot \mathbf{x}_0) = (\mathbf{n}_h, \mathcal{H}_h(\mathbf{x}_0))$. We like to show that given a collection of hyperplanes $\mathbf{h} = \{(\mathbf{n}_h, d_h)\}$, if we consider all possible translations of this collection, there is a unique one that minimizes

$$\mathcal{L}_{\text{unique}} = \frac{1}{H} \sum_h \mathcal{H}_h(\mathbf{x})^2.$$

In fact, we can rewrite this sum as

$$\begin{aligned} \sum_h (\mathcal{H}_h(\mathbf{x}))^2 &= \|(\mathcal{H}_h(\mathbf{x}))_h\|^2 \\ &= \|(d_h + \mathbf{n}_h \cdot \mathbf{x})_h\|^2 \\ &= \|\mathbf{d_h} + N\mathbf{x}\|^2 \end{aligned}$$

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

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

Observation 1. *Let $\mathcal{H}_h(\mathbf{x}) = \mathbf{n}_h \cdot \mathbf{x} + d_h$ be a hyperplane and let $\mathbf{x}_0$ be any point. Then the distance between $\mathbf{x}_0$ and the hyperplane $\mathcal{H}_h$ is given by*

$$\frac{|\mathcal{H}_h(\mathbf{x}_0)|}{\|\mathbf{n}_h\|} \tag{14}$$

Proof. Let X be any point on the hyperplane h, so $\mathcal{H}_h(X) = 0$. Then the distance of $\mathbf{x}_0$ to $\mathcal{H}_h$ is the norm of the projection of $X - \mathbf{x}_0$ onto the norm of $\mathcal{H}_h$, $\mathbf{n}_h$. Using standard formulas we get:

$$\begin{aligned} \|\operatorname{Proj}_{n_h}(X - \mathbf{x}_0)\| &= \|\tfrac{(\mathbf{x}_0 - X)\cdot \mathbf{n}_h}{\|\mathbf{n}_h\|^2} \mathbf{n}_h\| \\ &= \tfrac{\|\mathbf{n}_h \cdot x_0 - \mathbf{n}_h \cdot X\|}{\|\mathbf{n}_h\|} \\ &= \tfrac{\|\mathbf{n}_h \cdot x_0 + d_h\|}{\mathbf{n}_h\|} \end{aligned}$$

as desired. □

Note that if we assume that $\mathbf{n}_h$ are all normalized, then $|\mathcal{H}_h(\mathbf{x}_0)|$ gives us the distance to the hyperplane h for each hyperplane h. Therefore, given a convex polytope defined by collection of hyperplanes $\mathbf{h} = \{(\mathbf{n}_h, d_h)\}$, the $\sum_h |\mathcal{H}_h(\mathbf{x})|^2$ is the sum of squares of distance of $\mathbf{x}$ to the sides of the polytope.

Also note that if we translate $\mathbf{h}$ by $\mathbf{x}_0$, then we get the new set of hyperplane $\mathbf{h}' = \{(\mathbf{n}_h, d_h + \mathbf{n}_h \cdot \mathbf{x}_0)\} = \{(\mathbf{n}_h, \mathcal{H}_h(\mathbf{x}_0))$. Therefore,

$$\mathcal{L}_{\text{unique}} = \sum_h \|\mathcal{H}_h(\mathbf{0})\| \tag{15}$$

can be interpreted as the square distance of the origin to the collection of hyperplanes $\mathbf{h}$.