DOI: 10.1111/cgf.14020
EUROGRAPHICS 2020
R. Mantiuk and V. Sundstedt
(Guest Editors)

Volume 39 (2020), Number 2
STAR – State of The Art Report

Learning Generative Models of 3D Structures

Siddhartha Chaudhuri[1,2] Daniel Ritchie[3] Jiajun Wu[4] Kai Xu[5†] Hao Zhang[6]

[1]Adobe Research [2]IIT Bombay [3]Brown University [4]Stanford University
[5]National University of Defense Technology [6]Simon Fraser University

Figure 1: *A sampler of representative results from generative models of 3D shape and scene structures that were learned from data [HKM15, FRS*12, WLW*19, LXC*17].*

Abstract

3D models of objects and scenes are critical to many academic disciplines and industrial applications. Of particular interest is the emerging opportunity for 3D graphics to serve artificial intelligence: computer vision systems can benefit from synthetically-generated training data rendered from virtual 3D scenes, and robots can be trained to navigate in and interact with real-world environments by first acquiring skills in simulated ones. One of the most promising ways to achieve this is by learning and applying generative models *of 3D content: computer programs that can synthesize new 3D shapes and scenes. To allow users to edit and manipulate the synthesized 3D content to achieve their goals, the generative model should also be* structure-aware*: it should express 3D shapes and scenes using abstractions that allow manipulation of their high-level structure. This state-of-the-art report surveys historical work and recent progress on learning structure-aware generative models of 3D shapes and scenes. We present fundamental representations of 3D shape and scene geometry and structures, describe prominent methodologies including probabilistic models, deep generative models, program synthesis, and neural networks for structured data, and cover many recent methods for structure-aware synthesis of 3D shapes and indoor scenes.*

CCS Concepts
• ***Computing methodologies*** → *Structure-aware generative models; Representation of structured data; Deep learning; Neural networks; Shape and scene synthesis; Hierarchical models;*

† Corresponding author: kevin.kai.xu@gmail.com

1. Introduction

3D models of objects and scenes are critical to many industries, academic disciplines, and other applications. The entertainment industry—including games, animation, and visual effects—demands 3D content at increasingly large scales to create immersive virtual worlds. Virtual and augmented reality experiences, now becoming mainstream, also require this type of content. Architects, interior design firms, and furniture retailers increasingly rely on 3D models of objects, buildings, and indoor living spaces to visualize possibilities, advertise products, and otherwise engage with customers. Of particular interest is the emerging opportunity for 3D graphics to be a service for artificial intelligence: computer vision systems can benefit from synthetically-generated training data rendered from virtual 3D scenes, and robots can be trained to navigate in and interact with real-world environments by first acquiring skills in simulated ones.

Ideally, the stakeholders in these different areas would have access to 3D modeling tools which would allow them to easily create, edit, and manipulate 3D objects and scenes in order to satisfy their goals. Unfortunately, this ideal is far from being realized. Though the demand for 3D content has never been greater, the practice of creating 3D content is still largely inaccessible. Traditional 3D modeling software is notoriously complex, requiring extensive training and experience to be used at a high proficiency level. A potential alternative to modeling virtual 3D worlds "from scratch" is to instead acquire them from the real world, via photogrammetry or some other 3D scanning process, followed by 3D shape/scene reconstruction. Such processes come with their own problems, however: the scanning equipment can be expensive and cumbersome to use, the 3D models produced are often of poor quality and lack the organization and structure necessary to make them easily editable, and last but not the least, reconstruction from real-world scenes does not allow controllability over the generated content and limits the creativity of the content developer.

One of the most promising ways out of this quagmire is through *generative models* of 3D content: computer programs that can synthesize new 3D shapes and scenes. To be usable by non-expert 3D modelers in the disciplines mentioned previously, these programs should not require significant technical expertise to create—rather than being written by hand, they should be *learned from data*. To allow users to edit and manipulate the synthesized 3D content to achieve their goals, the generative model should also be *structure-aware*: it should express 3D shapes and scenes using abstractions that allow manipulation of their high-level structure.

This report surveys historical work and recent progress on learning structure-aware generative models of 3D shapes and scenes. We first discuss the different representations of 3D geometry and 3D structure that have been proposed, with their respective strengths and weaknesses. Next, we give an overview of the most prominent methods used to define and learn generative models of structured 3D content. We then survey the applications in which these ideas have been used, ranging from synthesis of part-based shapes to generation of indoor scenes. Finally, we highlight some remaining open problems and exciting areas for future work.

This report is a follow-up to our previous tutorial of the same name [CRXZ19]. We cover an expanded scope of material (including program induction methods), more comprehensively survey work in different sub-fields of 3D structure generation, and include references to the very latest, cutting-edge results. We also present a more systematic breakdown of different 3D structure representations, methods for generating them, and under what circumstances each method is likely to perform best.

2. Background & Scope

While there are many potential readers who would benefit from learning about the subject matter of this report, we have written it with new graduate students in computer science (and related disciplines) in mind as a target audience. Readers are expected to have a solid understanding of the fundamentals of linear algebra, probability and statistics, and computer graphics. Some prior exposure to machine learning concepts will also be helpful (e.g. training vs. test sets, overfitting), as will exposure to basic neural network concepts (multi-layer perceptrons, convolution, activation functions, stochastic gradient descent, etc.)

This report is also, by necessity, limited in scope. Specifically, we focus on *learned* generative models of *structured* 3D content. A *learned* model is one whose behavior is determined via a database of examples, rather than authored by hand or any other rule-based procedure. *Structured* 3D content refers to 3D shapes and scenes that are decomposed into sub-structures (e.g. 3D objects made up of an assembly of parts), as opposed to a monolithic chunk of geometry. There are many areas of research related to our topic of focus which we do not discuss in detail. These include:

Non-learned generative models of structured 3D content. The idea of creating computer programs to synthesize structured 3D content is an idea almost as old as computer graphics itself. Grammar-based methods, i.e. recursive expression-rewriting systems, are the most well-established and most popular. Lindenmayer systems (L-systems) are a type of grammar in which repeated expression rewriting is interpreted as gradually growing a structure. L-systems were popularized by their application to generating trees and other vegetation [PL96]. Shape grammars are another popular type of grammar; here, expression rewriting is interpreted as repeatedly subdividing an initial shape / region of space to produce geometry. These have been used for generating buildings [MWH*06], building facades [MZWVG07], and complete cities [PM01]. The above methods all directly generate content by forward execution of some procedure. There is also significant prior work on methods that indirectly generate content via optimization, e.g. optimizing the parameters of a procedure to produce desired outputs. Such examples include optimizing the output of grammars [TLL*11], more general object modeling programs [RMGH15], and procedures that produce object layouts [YYT*11, MSL*11, YYW*12].

Learned generative models of non-structured 3D content. Recent advances in deep learning have enabled synthesis of complex 3D shapes and scenes without explicit modeling of their internal structure. Wu et al. [WSK*15] first used a Deep Belief Network (DBN) to model a probabilistic space of 3D shapes, and demonstrated that such a model can be used to synthesize

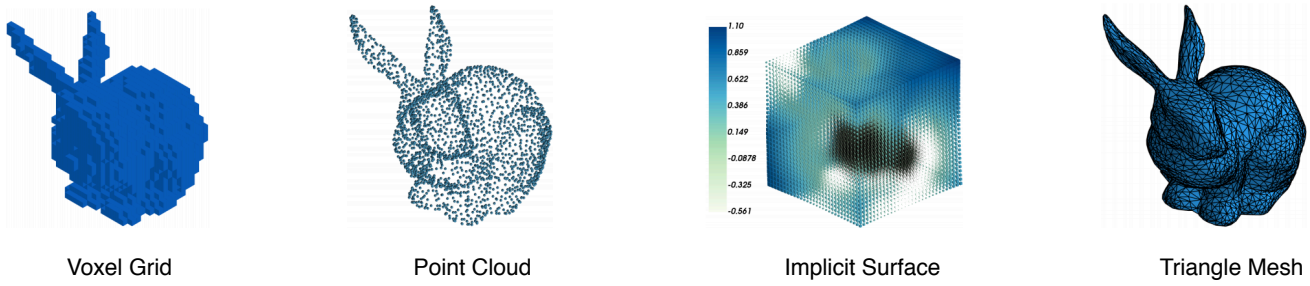


Figure 2: *Different representations for the low-level geometry of individual objects / parts of objects [MPJ*19].*

novel 3D shapes after training on large shape repositories such as ShapeNet [CFG*15]. Girdhar et al. [GFRG16] later proposed to connect the latent representations of 3D shapes and 2D images for single-image 3D reconstruction. Sharma et al. [SGF16] explored using denoising autoencoders to learn a latent space of 3D shapes for shape completion.

Other popular deep learning methods have also been applied to 3D content synthesis. Wu et al. [WZX*16] first applied generative adversarial networks (GAN) [GPAM*14] to voxels; their 3D-GAN was able to first perform unconditioned synthesis of high-resolution 3D shapes. Liu et al. [LYF17] extended the model to allow interactive shape synthesis and editing. As discussed later in Section 3.1, GANs have later been adapted to other shape representations, achieving great successes with point clouds, octrees, and surface meshes.

Variational autoencoders (VAEs) [KW14] have also demonstrated their potential in unconditional 3D shape synthesis [BLRW16, SHW*17], especially on domain-specific areas such as face and human body modeling with meshes [RBSB18, TGLX18]. With these modern deep learning tools, researchers have been able to obtain very impressive results on shape synthesis and reconstruction.

These papers mostly ignore the structure of 3D shapes and scenes, and instead aim to synthesize their entire geometry "from scratch." In this survey, we focus on works which learn to synthesize 3D content while explicitly incorporating their internal structure as priors.

3. Structure-Aware Representations

A *structure-aware* representation of a 3D entity (i.e. a shape or scene) must contain at least two subcomponents. First, it must have some way of representing the geometry of the atomic structural elements of the entity (e.g. the low-level parts of a 3D object). Second, it must have some way of representing the structural patterns by which these atoms are combined to produce a complete shape or scene. In this section, we survey the most commonly-used representations for both of these components.

3.1. Representations of Part/Object Geometry

In computer graphics, 3D geometric objects are mainly represented with voxel grids, point clouds, implicit functions (level sets), and surface meshes (see Figure 2). Other alternatives include parametric (e.g., tensor-product) surfaces, multi-view images [SMKL15], geometry images [GGH02], and constructed solid geometry. For the purpose of shape generation, voxel grids are the simplest representation to work with: a 3D voxel grid is a straightforward extension of a 2D pixel grid, making it easy to transfer deep convolution-based models developed for 2D image synthesis to the 3D domain [WSK*15, WZX*16]. However, it is difficult to generate high-detail shapes with such volumetric representations, as the memory cost of the representation scales cubically with the resolution of the voxel grid. This problem can be partially alleviated by using efficient and adaptive volumetric data structures such as octrees [ROUG17, WLG*17, KL17, TDB17].

With the proliferation of neural networks for point cloud processing [QSMG17, QYSG17], deep generative models for direct synthesis of 3D point clouds have become popular. A point cloud represents only the surface of a 3D object and thus sidesteps the resolution restriction of volumetric representations [FSG17, ADMG18]. However, point clouds are only discrete samples of a surface and do not provide information about the continuous behavior of the surface between point samples.

A compelling alternative to voxel and point based representations are implicit representations: functions $f(x,y,z)$ whose output determines whether a point is inside or outside of a surface. Such a function captures the smooth, continuous behavior of a surface and enables sampling the implied surface at arbitrary resolution. Recent efforts have shown that it is possible to use neural networks to learn such functions from data [CZ19, PFS*19, MPJ*19, MON*19]. At present, these methods are the state-of-the-art approaches for generating *unstructured*, detailed geometry.

Another possible approach is to consider directly generating a triangle mesh representation of a surface. This is an appealing idea, as meshes are the most commonly used geometry representation for most graphics tasks. An early step in this direction generated "papier-mache" objects by learning to deform and place multiple regular mesh patches [GFK*18, YFST18]. Alternatively, one can define a convolution operator on meshes [MBBV15, HHF*19] and attempt to generate meshes by iterative subdivision + convolution starting from some simple initial mesh. This approach has been used for 3D reconstruction from images [WZL*18, GG19] and from 3D scans [DN19].

Figure 3: *Representing 3D structure via segmented geometry.* Left: *representing the part structure of a 3D object by part-labeled voxels [WSH*18].* Right: *Representing a 3D scene via a floor plan image with per-pixel object type labels [WSCR18].*

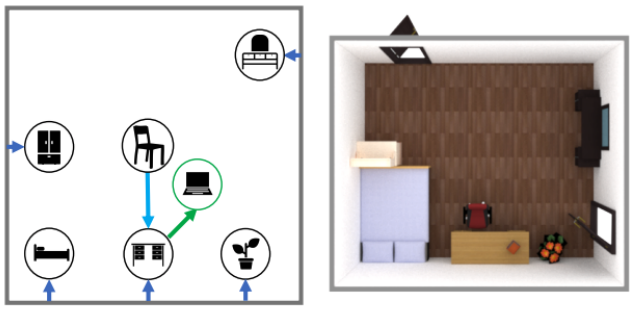


Figure 4: *Representing 3D structures via relationship graphs [WLW*19]. Edges between object nodes for this 3D scene graph indicate spatial proximity and physical support.*

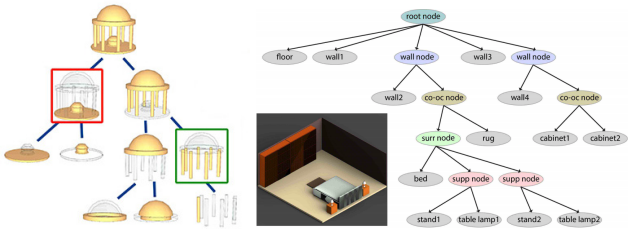


Figure 5: *Representing 3D structures via hierarchies.* Left: *a symmetry hierarchy for a 3D architectural shape, formed via repeatedly contracting the edges of an adjacency + symmetry relationship graph [WXL*11].* Right: *a 3D bedroom scene represented via a hierarchical grouping of objects [LPX*19].*

3.2. Representations of Structure

The previous section outlined the most frequently used representations for the individual atomic parts of objects (or the objects within larger scenes). In this section, we survey a range of ways to represent 3D structures composed of atomic parts (or objects), in increasing order of structural complexity.

Segmented geometry. Perhaps the simplest way to add structure to a representation of a 3D entity is to simply associate structural labels to each part of the entity's geometry. For example, a voxel-based representation of a 3D object can become "structured" via associating a part label with each voxel [LNX19, WSH*18], or a surface can be represented by a set of primitive patches [GFK*18, DGF*19]. Analogously, a 3D scene can be represented via a top-down 'floor plan' image, where each pixel in the image denotes the type of entity present at that location [WSCR18]. Figure 3 shows examples of both of these settings. This representation is simple to construct, and carries the advantage that it is easy to analyze and synthesize via the same machine learning models which operate on the corresponding unlabeled geometry. However, as the individual atomic parts/objects are only implicitly defined (e.g. by connected components of voxels with the same label), machine learning models which are trained to generate them must learn to identify and recreate these atoms (or suffer from visual artifacts in their generated output). In contrast, representations which explicitly denote individual structural atoms side-step this issue.

Part sets. The simplest *explicit* representation of structural atoms is an unordered set of atoms. In such a representation, each atomic part or object is typically associated with information about the atom's spatial configuration (e.g. an affine transformation matrix); it may also carry a type label [SSK*17]. These representations are also typically paired with a part geometry representation from Section 3.1 to represent the geometry of individual parts; machine learning models which generate such structures may use separately-trained components to handle the part set generation vs. the part geometry generation (which makes these models more complex than those which generate segmented geometry). This representation also does not encode any relationships between the structural atoms: rather, they are 'free-floating' objects in space.

Relationship graphs. An obvious extension of the part set representation is to convert the set of parts into a graph by inserting edges between parts denoting pairwise relationships between them. For 3D objects, these edges typically indicate either physical adjecencies between parts or higher-level spatial relations such as symmetries. For 3D scenes, edges can additionally denote semantic and functional relationships such as one object physically supporting another [FSH11, WLW*19]. Figure 4 shows a 3D scene represented in the relationship graph form. The structural relationships encoded by such scene graphs can then be characterized by descriptors, such as graph kernels [FSH11], for scene comparison. One can also analyze a collection of scene graphs to extract useful information such as *focal points* [XMZ*14], which are representative sub-scenes for a given scene collection. These focal points enable part-in-whole sub-scene retrievals and scene exploration.

While attractive for their generality and representation flexibility, building effective generative models of arbitrary graphs is at present still a challenging problem and the subject of much ongoing interest in the broader machine learning community [LVD*18, YYR*18, GZE19].

Hierarchies. One can derive many of the benefits of a graph-based structure representation, while also reducing the difficulty of learning from such data, by working with a restricted class of graphs. Trees are a natural choice, as many 3D structures observed in the

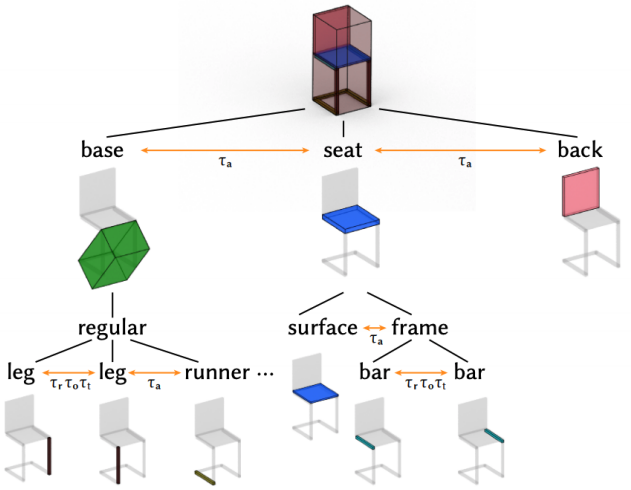


Figure 6: *Representing a chair via a hierarchical graph [MGY*19a]. Hierarchy levels encode coarse-to-fine structural grouping, while edges between nodes at the same hierarchy level encode adjacency and symmetry relationships.*

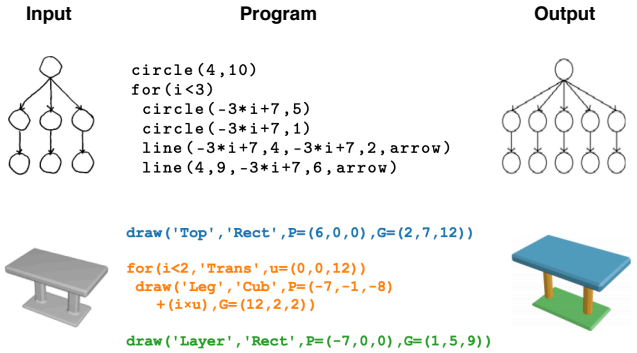


Figure 7: *Representing structures via programs that generate them.* Top: *a 2D structure represented via a program is 'extrapolated' via editing a loop in the program [ERSLT18].* Bottom: *a 3D chair represented by a simple shape program [TLS*19].*

real world exhibit hierarchical organizational patterns. For example, a chair contains a back part, which itself may be decomposed into subparts. Also, a 3D scene may be interpreted as a collection of "functional groups" of objects which decompose into coherent arrangements of objects (e,g. many bedrooms consist of a "bed-and-nightstands" furniture group, as depicted in Figure 5 Right).

One particularly noteworthy type of hierarchy is the *symmetry hierarchy*, which is a tree organization of 3D shape parts [WXL*11]. A symmetry hierarchy is a binary tree whose leaf nodes correspond to parts, and whose internal nodes are formed by repeatedly merging two nodes which are related via an adjacency or symmetry relationships; see Figure 5 (left) for an example on an architectural object. Interestingly, the construction of a symmetry hierarchy can be viewed as a process of repeatedly contracting the edges of a relationship graph until only a single (root) node remains. This mapping from relationship graph to symmetry hierarchy is not unique, however: there are many possible hierarchies for each graph, depending on the order of edge contraction chosen.

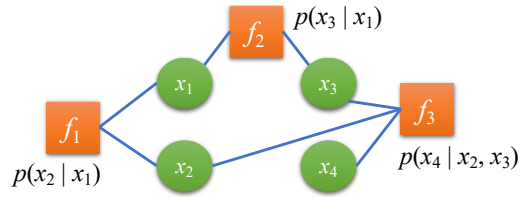
Hierarchical graphs. An even more flexible representation for 3D structure comes from combining the representational advantages of both hierarchies and graphs in the form of hierarchical graphs. A hierarchical graph is a graph in which each node can itself expand into another hierarchical graph, terminating at nodes which represent atomic geometry. Alternatively, it can be viewed as a graph in which sub-graph nodes are connected to their hierarchical parent node via special 'parent edges.' This representation is able to represent the hierarchical grouping patterns observed in real-world shapes and scenes while also benefiting from undirected edges at each hierarchy level to represent spatial or functional relationships. While a powerful representation, the machine learning models needed to generate hierarchical graphs can be very complex, and the data needed to train such models (3D objects/scenes with complete grouping and relationship annotations) can be hard to come by or expensive to produce. Figure 6 shows the hierarchical graph representation for a 3D chair model.

Programs. Perhaps the most general way to represent a spatial structure is via a deterministic program which outputs that structure. In some sense, this representation subsumes all the others in this section, in that a program can be made to output any of the previously-discussed representations. Programs have the advantage of making patterns and dependencies between structural elements clear, and expressing them in a way that allows easy editing by a user (e.g. changing the number of repetitions of a pattern by editing a for loop). Figure 7 shows examples of program representations used to represent different structures.

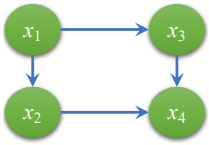
4. Methods

In this section, we cover prominent methodologies for learning structured models that are applicable to shape and scene generation. A *generative* model, strictly speaking, represents a (conditional or unconditional) joint probability distribution over an input space X (e.g., a shape space) and hence can be used to sample (i.e., synthesize) objects from X. This is in contrast to a *discriminative* model which only represents the distribution of some attribute $y \in Y$ (e.g., a class label) conditioned on a given object $x \in X$. Nevertheless, the term "generative" is sometimes loosely applied to any synthesis method. In this survey, we focus primarily on strictly generative models, but some cited works may deviate from that rule or come with caveats.

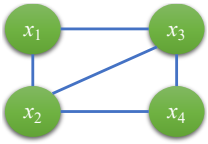
Early generative methods were mostly based on probabilistic models which generate outputs by sampling an estimated probabilistic distribution. Then the "era of deep learning" popularized the use of deep neural networks for generative modeling. While most deep generative networks such variational autoencoders and adversarial networks are generic models, there are networks which are specifically tailored to learning tree and graph structures, e.g., recursive neural networks and graph convolutional networks. We conclude our coverage with methods that synthesize program-



(a) *Factor graph*



(b) *Bayesian network*



(c) *Markov random field*

Figure 8: *Three flavors of probabilistic graphical models can all represent the same distribution:* $p(\mathcal{X}) = p(x_4 \mid x_2, x_3) \times p(x_3 \mid x_1) \times p(x_2 \mid x_1) \times p(x_1)$.

based representations, including how the adoption of neural networks has shape recent developments along this direction.

4.1. Classical Probabilistic Models

A variety of trainable generative models were developed to represent probability distributions that possess a structure-based factorization. We discuss the most important ones for structure-aware shape generation in this section.

Probabilistic graphical models. Graphical models are an elegant method to represent a family of objects as a joint probability distribution $p(\mathcal{X})$ over discrete or continuous-valued object properties (variables) $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$. While the complete distribution may have a complex mass function, it can often be compactly represented in a *factorized* form over subsets of variables:

$$p(\mathcal{X}) = \prod_{i=1,\, X_i \subseteq \mathcal{X}}^{m} f_i(X_i).$$

Each $f_i(X_i)$ is called a *factor*, and is typically a simple conditional probability function correlating the variables in X_i. A graphical model uses a *graph* (hence the name) $G = (V, E)$ to visually represent the factorization. There are three principal flavors:

Factor graph: The most general form, with vertices $V = \mathcal{X} \cup \mathcal{F}$, where $\mathcal{F} = \cup_i \{f_i\}$. The graph is bipartite, with undirected edges connecting the variables X_i of the i^{th} factor to the factor node f_i.

Bayesian network: A directed ayclic graph with vertices $V = \mathcal{X}$. The joint distribution factorizes over the vertices as

$$p(\mathcal{X}) = \prod_{x_i \in \mathcal{X}} p\left(x_i \mid \text{Parents}(x_i)\right),$$

where $\text{Parents}(x_i)$ are the parent nodes of x_i in the graph. Each variable is independent of all others, given its parents.

Markov random field (MRF): An undirected graph with vertices $V = \mathcal{X}$, usually representing a distribution where each variable is independent of all others, given its neighbors. By the Hammersley-Clifford Theorem, the probability function, if everywhere positive, factorizes over maximal cliques of the graph.

START : ***F***
F ⟶ [Left] ***F*** (0.2)
F ⟶ [Right] ***F*** (0.2)
F ⟶ [Left] ***F*** ∪ [Right] ***F*** (0.3)
F ⟶ [Leaf] (0.3)

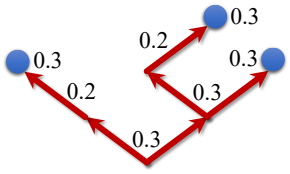


Figure 9: *A simple non-parametric PCFG that generates branching structures. The grammar has one nonterminal: F; three terminals: a leftward arrow, a rightward arrow, and a circle; and four rules with the indicated probabilities. The probability of the derived tree is the product of the applied rule probabilities:* $p(\mathcal{T}) = 0.3 \times 0.2 \times 0.3 \times 0.2 \times 0.3 \times 0.3 \times 0.3$.

The above representations are completed by specifying every factor's probability function f_i, as either a table (for discrete variables) or a continuous model. Any Bayes net or MRF can be expressed as a factor graph. Bayes nets can often be converted to MRFs and vice versa. Bayes nets are typically used for probabilistic systems where there are directed/causal relationships, whereas MRFs are appropriate for undirected, energy-driven systems. Figure 8 shows three representations of the same distribution.

Various algorithms exist for learning both the graph structure and factor parameters from training data [KF09]. For instance, structure can be learned via randomized search, and parameters by expectation-maximization for each candidate structure. Note that some variables may be internal to the model and not directly observed in the data: these are called "latent" or "hidden" variables, and they help mediate correlations. There are also several ways to sample the learned distribution, and infer conditional distributions of query variables given observations of other variables.

For the specific purpose of shape synthesis, the variables in a graphical model typically represent properties of semantic components of the shape family, e.g. the seat, arms, legs and back of a chair. The model is trained on a repository of existing shapes, and new shapes can be sampled from it, optionally constrained on known properties. The model can also be used as a generative prior for other objectives such as language or image-driven synthesis.

The principal limitations of traditional graphical models are that they require careful structure and feature engineering, do not scale well to complex high-dimensional (or variable-dimensional) distributions, and training and inference can be difficult. Hence, modern shape generation has largely moved to deep neural networks and recurrent/recursive models. Nevertheless, with proper care, graphical models can be powerful, (relatively) interpretable high-level priors, especially for smaller, well-parametrized datasets.

Probabilistic grammars. Context-free grammars (CFGs) are hierarchical models which repeatedly apply *production rules* to parts of an object, typically to replace a simple part with a more complex pattern. Different parts may be expanded independently, giving derivations from a grammar their characteristic branching structure. Formally, a CFG is a 4-tuple (Σ, V, R, s), where Σ is a set of *terminal* symbols, V is a set of *nonterminal* symbols, R is a set

of *rules* where each rule maps a nonterminal to a layout of terminals/nonterminals, and $s \in V$ is a *start* symbol. A derivation from the grammar is a sequence of rules that transforms the start symbol into an output pattern by successively replacing a selected nonterminal matching the LHS of a rule with the RHS of the rule, until the output consists only of terminals.

A *probabilistic* context-free grammar (PCFG) augments this setup with a probability for each rule. Such a model is generative: it defines a probability distribution over derivations. If the derivation $\mathcal{T}$ is the sequence $[(r_1, s_1 = s), \ldots, (r_n, s_n)]$, where rule r_i is applied to symbol s_i in each step, then the probability of the derivation is simply $p(\mathcal{T}) = \prod_i^n p(r_i)$. Figure 9 shows an example of a derivation from a toy PCFG that generates branching structures. A more general variant is a *parametric* PCFG, where each symbol has associated parameters (for example, the size of the corresponding component), and the application of a rule involves sampling the parameters of the RHS conditioned on the parameters of the LHS. Thus, if step i replaces symbol s_i with symbols $c_1 \ldots c_k$, then

$$p(\mathcal{T}) = \prod_i^n p\left(c_1 \ldots c_k \mid \phi_i(r_i, s_i)\right),$$

where ϕ_i determines the distribution from which the parameters of the output symbols are drawn. This expression explicitly highlights the Markovian nature of the derivation: each step in the derivation is conditioned only on the previous one, and more specifically on just one symbol – the "parent" – in the intermediate pattern. PCFGs can be thought of as "dynamic" Bayesian networks, whose output is variable-dimensional. This gives them richer expressive power than the "static" graphical models described above.

The dynamic, recursive nature of PCFGs makes them well-suited for model structures that are defined by recursive or hierarchical processes. Trees and other types of vegetation are the most common examples, but buildings and other shape structures that feature hierarchies of spatial patterns are also good candidates. Typically, PCFGs model shapes by treating pieces of atomic geometry (i.e. shape parts) as terminal symbols. Since their derivation process is tree-structured by nature, PCFGs can be difficult to adapt to shapes that do not exhibit tree-like structures (e.g. part-based shapes whose connectivity graphs include cycles).

While PCFG rules are typically designed by hand, and probabilities and other parameters estimated from data, an appropriate set of rules can also be learned from examples. This is known as grammar induction, and is a distinctly non-trivial exercise. Examples from shape generation are rare: some are described below.

Probabilistic programs. In the study of the theory of computation, context-free grammars (CFGs) are capable of recognizing any context-free language, whereas a Turing machine can recognize any language (and thus a language which requires a Turing machine to be recognized is said to be Turing-complete). Probabilistic programs are to *probabilistic* context-free grammars (PCFGs) what Turing machines are to deterministic CFGs: where PCFGs can represent only a certain class of probability distributions over tree structures, probabilistic programs can represent *any computable probability distribution* [GMR*08]. In particular, any PCFG can be represented by a probabilistic program.

Practically, probabilistic programs are usually implemented by extending existing deterministic Turing-complete programming languages (e.g. Lisp [GMR*08], Javascript [GS14], C [PW14], Matlab [WSG11]) with primitive operators for sampling from elementary probability distributions (e.g. Gaussian, Bernoulli). Executing a probabilistic program produces a *trace* through the program, which is a sequence of all the random sampling decisions made by the program (analogous to a derivation from a PCFG). The probability of a particular trace $\mathcal{T}$ is

$$p(\mathcal{T}) = \prod_{\mathbf{x}_i \in \mathcal{T}} p(\mathbf{x}_i | \phi_i(\mathbf{x}_1 \ldots \mathbf{x}_{i-1})),$$

where $\mathbf{x}_i$ is the i^{th} random choice in the trace and $\phi_i(\mathbf{x}_1 \ldots \mathbf{x}_{i-1})$ encapsulates whatever computation the program performs to compute the parameters of the i^{th} random choice (e.g. mean and variance, if it is a sample from a Gaussian distribution). Expressed in this form, it is clear how probablistic programs subsume PCFGs: if the ϕ function retrieves data from a parent in a tree structure, then this equation can be made to exactly express the probability distribution defined by a PCFG.

Probabilistic programs show their true power as a representation for generative models with their capabilities for *inference*: most probabilistic programming languages come equipped with built-in operators for sampling conditional probability distributions of the form

$$p(\mathcal{T} | c(\mathcal{T}) = \text{true}).$$

That is, they provide the ability to compute how the distribution of execution traces changes when some condition c on the trace is required to be true (e.g. that certain random choices take on specific values). Computing these kinds of conditional distributions is intractable in general, so real-world probabilistic programming languages make sacrifices, either by restricting the types of conditions c that are expressable and/or by only computing approximations to the true conditional distribution.

Whereas PCFGs are restricted to generating tree structures, probabilistic programs can in theory generate any shape structure (e.g. cyclically-connected arches [RLGH15]). This generality comes at a cost in terms of learning, though. While it is difficult to induce a PCFG from data, some approaches do exist. However, there is no known general-purpose algorithm for inducing probabilistic programs from examples. In the domain of shape synthesis, some domain-specific approaches have been proposed, though the kinds of programs they induce are not much more expressive than grammars [HSG11, RJT18].

4.2. Deep Generative Models

The generative models surveyed in the last section were all developed before the deep learning "revolution" which began in approximately 2012. Since that time, new types of generative models with deep neural networks at their core have been developed—so-called *deep generative models*. In some cases, these models actually have close connections to the "classical" models we have already discussed; we will highlight these connections where they exist.

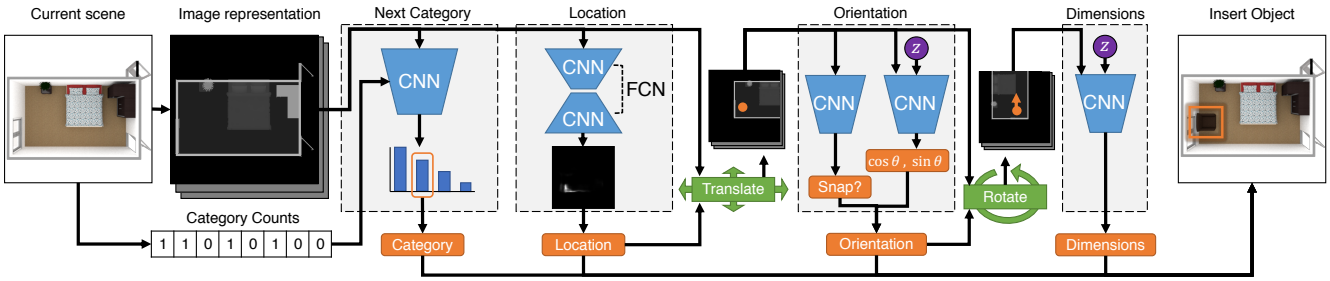


Figure 10: *An autoregressive model for generating 3D indoor scenes [RWaL19]. The model generates scenes one object at a time (and one attribute of each object at a time), conditioned on the partial scene that has been synthesized thus far.*

Autoregressive models. An *autoregressive* model is an iterative generative model that consumes its own output from one iteration as the input to its next iteration. That is, an autoregressive model examines the output it has generated thus far in order to decide what to generate next. For example, an autoregressive model for generating 3D indoor scenes could synthesize one object at a time, where the decision about what type of object to add and where to place it are based on the partial scene that has been generated so far (Figure 10). Formally, an autoregressive model defines a probability distribution over vectors $\mathbf{x}$ as

$$p(\mathbf{x}) = \prod_{i=1}^{|\mathbf{x}|} p(\mathbf{x}_i | \mathrm{NN}_i(\mathbf{x}_1 \ldots \mathbf{x}_{i-1})),$$

where NN_i denotes a neural network. In other words, the model represents a distribution over (potentially high-dimensional) vectors via a product of distributions over each of its entries. In the process of sampling from this distribution, the value for $\mathbf{x}_{i-1}$ must be fed into the neural network to sample a value for $\mathbf{x}_i$, which earns this type of model its autoregressive name.

If NN_i is allowed be a different neural network at each step, this representation is very similar to a probabilistic program, where the parameter function ϕ is a learnable neural network instead of a fixed function. More commonly, the same neural network is used at every step (e.g. recurrent neural networks are a form of autoregressive generative model that fits this description, as are pixel-by-pixel image generative models [VDOKK16]).

Autoregressive models are well-suited to generating linear structures (i.e. chains), as well as for mimicking structure design processes that involve making a sequence of decisions (e.g. placing the objects in a scene one after another). Their primary weakness is their susceptibility to "drift": if an output of one step differs from the training data in some statistically noticeable way, then when fed back into the model as input for the next step, it can cause the subsequent output to diverge further. Such errors can compound over multiple prediction steps.

Deep latent variable models. Recently, two generative frameworks have become extremely popular in deep neural network research: variational autoencoders (VAEs) and generative adversarial networks (GANs). Both these methods allow us to efficiently sample complex, high-dimensional probability distributions, such as that specified by a plausibility prior over shapes, images or sentences. This is a classic hard problem, traditionally addressed with slow, iterative methods like Markov Chain Monte Carlo. Under the right conditions, if the distribution is represented by a set of landmarks (i.e. an unsupervised training set), sampling can be reduced to just a forward pass through a neural network, conditioned on random noise from a known distribution.

The core idea behind both VAEs and GANs is the following: let $p : X \to \mathbb{R}$ be the probability function for some hard-to-sample distribution over (say) 3D shapes X. We will sample instead from some "easy" distribution (typically the standard Gaussian) over a low-dimensional *latent space* Z, and learn a *generator* function $G : Z \to X$ that maps latent vectors to actual objects. Training tries to make $p(\mathbf{z}) = p(G(\mathbf{z}))$ for all $\mathbf{z} \in Z$, so sampling from the latent space, followed by decoding with the generator, is equivalent to sampling from the target distribution. VAEs and GANs are different methods to learn a good generator G.

A VAE [KW14] can be thought of as maximizing a variational Bayes objective $p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \mathrm{d}\mathbf{z}$ over all training examples $\mathbf{x} \in X$, where $p(\mathbf{x} \mid \mathbf{z})$ is modeled by the generator. We desire $p(\mathbf{x} \mid \mathbf{z})$ to be smooth and compact (a single latent should generate similar outputs): this is achieved indirectly by making $p(\mathbf{z} \mid \mathbf{x})$ smooth and compact (a single output can be traced back to a small range of similar latents). A VAE enforces this with an *encoder* neural network that maps a training input $\mathbf{x}$ to a small Gaussian $\mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x}))$ in latent space. Samples from this Gaussian are fed to the generator, also modeled with a neural network, and the final output is constrained to be identical to the input. Thus, a VAE resembles an autoencoder where the bottleneck produces a compact distribution rather than a single latent vector. Since the prior distribution of latents is assumed to be (typically) the standard normal, we also constrain the mixture of Gaussians $\sum_{\mathbf{x} \in X} \mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x}))$ to match the prior. At test time, we simply sample the standard normal in latent space and decode the sampled vector to an actual object using the generator. Figure 11 illustrates the framework.

A GAN [GPAM*14] dispenses with the encoder network and instead uses the training exemplars to develop a "learned loss function", modeled by a *discriminator* network D. The generator tries to map latent vectors sampled from the standard normal prior to plausible objects. The discriminator tries to distinguish "fake" objects produced by the generator from "real" objects from the training set. This constitutes the adversarial nature of the framework, with each network trying to outdo the

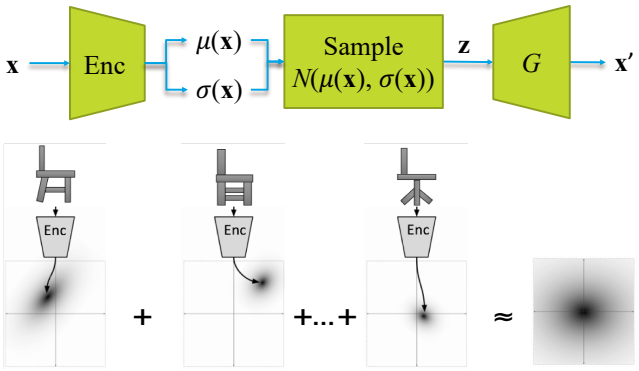


Figure 11: *A variational autoencoder (top) has an encoder network that maps a training exemplar to a small Gaussian in latent space, and a generator that maps a sample from this Gaussian back to the exemplar. The mixture of the per-exemplar Gaussians is constrained to be the standard normal distribution (bottom). At test time, sampling from the standard normal and decoding with the generator has the effect of sampling the probabilistic manifold represented by the training set.*

other. As the generator gets better at fooling the discriminator, the discriminator also gets better at distinguishing fake from real. Training minimizes over G, and maximizes over D, the loss $\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$.

Both VAEs and GANs can be conditioned on additional input, to enable synthesis guided by user actions or data in other modalities. They are suitable for modeling a wide variety of shape structures, depending upon the specific architectures chosen for their decoder/generator networks (see the next section for more on this). The use of a global latent variable to control the generation of the structure encourages global coherence in the output, in contrast to the tendency of autoregressive models to drift (i.e. to generate locally-plausible but globally-incoherent structures). It is also possible to combine latent variable and autoregressive models, as an autoregressive model such as a recurrent neural network can be used as the decoder/generator of a VAE/GAN.

4.3. Neural Networks for Structured Data

The deep generative models defined in the last section used neural networks, which we denoted as "NN," as critical building blocks. The precise form (or *architecture*, in neural network parlance) that these networks take depends upon the type of data being modeled. In this section, we survey the types of neural network architectures that are most appropriate for the structured data representations discussed in Section 3.2.

NNs on trees: recursive neural networks. The layout of parts of a shape inherently induces a non-Euclidean, graph-based topology defined by inter-part relations. The main challenge in handling such graph-based data with deep NN is the non-uniform data format: the number of parts is arbitrary even for shapes belonging to the same category. In the field of natural language processing, researchers adopt a special kind of NN, named Recursive Neural Networks (RvNN), to encode and decode sentences with arbitrary lengths [SLNM11]. The premise is that a sentence can be represented with a parse tree, following which the word vectors can be aggregated and expanded in a recursive manner, up and down the tree. The general form of an RvNN is recursive bottom-up merging of tree nodes for encoding:

$$x_p = \tanh(W_e \cdot [x_l\ x_r] + b_e),$$

and top-down splitting for decoding:

$$[x_l\ x_r] = \tanh(W_e \cdot x_p + b_e),$$

where x_p, x_l, x_r represent the feature vectors of a parent node and its left and right children, respectively. $W_{ad} \in \mathbb{R}^{2n\times n}$ and $b_{ad} \in \mathbb{R}^{2n}$ are network parameters to be learned.

An RvNN encoder repeatedly applies the above merging operation to build a parse tree that collapses the variable sized input into a single root node with a fixed-dimensional feature vector. Similarly, an RvNN decoder recursively decodes the root vector back to constituent leaf nodes. Powerful generative models such as VAEs and GANs can be trained on these fixed-dimensional feature vectors, which are then decoded to actual, variable-dimensional objects.

More complicated encoder/decoder operations have been studied [SPW*13], but the idea of recursively using exactly the same NN module for merging (and similarly for splitting) remains the same. While RvNNs were originally applied to sentences, they can be applied to arbitrary graphs (including part layouts and scene graphs), where a merge corresponds to an edge collapse.

RvNNs permit both supervised (with labeled parse trees) and unsupervised (with just a reconstruction loss) training. At test time, the tree topology need not be part of the input but can be determined on the fly, e.g. by examining the reconstruction loss of a candidate grouping over one or more merge/split steps. Thus, RvNNs can be an alternative to the convolutional/message-passing networks that process arbitrary graph inputs, discussed below.

NNs on graphs: graph convolutional networks. In recent years, much research effort has been devoted to the application of neural network models to arbitrarily structured graphs. The most prevalent approach has been to generalize the concept of convolution from regular lattices to irregular graphs, which allows the construction of *graph convolutional networks* (GCNs). While some approaches to defining convolution on graphs are domain-specific [DMI*15, HHF*19], the most popular approaches are general-purpose and fall into one of two camps: *spectral graph convolution* or *message passing* neural networks.

Spectral graph convolution approaches define convolutional networks on graphs by approximating the equations for local spectral filters on graphs (i.e. multiplying a per-node signal by a filter in the frequency domain) [HBL15, KW17]. Just as image-based CNNs compute a sequence of per-pixel learnable feature maps, spectral GCNs compute a sequence of per-node learnable feature maps $\mathbf{H}^{(l)}$ via update rules like the following:

$$f(\mathbf{H}^{(l+1)}) = \sigma\left(\mathbf{A}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right),$$

where $\mathbf{W}^{(l)}$ is a weight matrix for the l-th neural network layer, $\sigma(\cdot)$ is a non-linear activation function, and $\mathbf{A}$ is related to the graph's Laplacian matrix [KW17].

Recently, message passing approaches (sometimes called message passing neural networks or MPNNs) have become more widespread than spectral approaches. Rather than operating in the frequency domain, MPNNs define convolution in the spatial domain by direct analogy to spatial convolution on images: the output of a convolution $f(\mathbf{h}_v)$ on some signal $\mathbf{h}_v$ at graph vertex v is some learned combination of the signal values at neighboring vertices. For a more detailed explanation, see Wang et. al [WLW*19], which adopts MPNNs + an autoregressive generative modeling approach for synthesizing indoor scene relationship graphs.

4.4. Program Synthesis

Finally, we highlight techniques for generating the program-based representations discussed at the end of Section 3.2. These techniques belong to the field of *program synthesis*, which is a much broader field of study that originated with and includes applications to domains outside of computer graphics (e.g. synthesizing spreadsheet programs from examples [Gul11]). As mentioned in Section 3.2, within the domain of shape synthesis, programs are well-suited to modeling any shape structure, as a program can be written to output many types of structure (linear chains, trees, graphs, etc.)

Constraint-based program synthesis. Program synthesis is a field with a lengthy history, with many different techniques proposed based on search, enumeration, random sampling, etc. [GPS17] Of late, the most prominent of these methods is *constraint-based* synthesis. In this formulation, the user first specifies the grammar of a *domain-specific language*, or DSL, which is the language in which synthesized programs will be expressed (for 3D shapes, this might be constructive solid geometry or some related CAD language). This DSL defines the space of possible programs that can be synthesized. Constraint-based program synthesis then attempts to solve the following optimization problem:

$$\begin{aligned} &\text{argmin}_{\mathcal{P}\in\text{DSL}}\text{cost}(\mathcal{P}), \\ &\text{s.t. } c(\mathcal{P}) = 0, \end{aligned}$$

where $\mathcal{P}$ is a program expressible in the DSL and c is some constraint function. In other words, the synthesizer tries to find the minimum-cost program which is consistent with some user-provided constraints. The constraint function c can take many forms; it is most commonly used to enforce consistency with a set of user-provided (input, output) example pairs (i.e. programming by demonstration). Cost is often defined to be proportional to program length / complexity, so as to encourage simpler programs that are more likely to generalize to unseen inputs (via an appeal to Occam's razor).

The optimization problem defined above is an intractable combinatorial problem, and the most challenging part of the problem is merely finding *any* program which satisfies the constraints. One approach is to convert this search problem into a Boolean satisfiability (SAT) problem, and then apply a state-of-the-art SAT solver [DMB08]. The Sketch system is perhaps the most well-known instance of this kind of constraint-based program synthesizer [Lez08]. While such solvers cannot always return a solution in a reasonable amount of time (in the worst case, searching for a satisfying program has exponential time complexity), they often perform well in practice, and they guarantee that the program they return is in fact the globally-optimal (i.e. minimum-cost) program.

Neural program synthesis. Beyond traditional program synthesis algorithms that rely mostly on hand-crafted constraints, neural nets can also be incorporated for more effective and efficient synthesis.

There are at least two ways that neural nets can be helpful. First, they can be used to speed up search-based synthesis by suggesting which search sub-trees to explore first, or whether a sub-tree should be explored at all [KMP*18, LMTW19]. Similarly, they may also be used to suggest parameters/attributes of function calls or programs [BGB*17, ERSLT18], to suggest program sketches [Lez08, MQCJ18], to include guidance from program execution [CLS19, TLS*19, ZW18, WTB*18], or to induce subroutines from existing programs [EMSM*18]. The main intuition behind these approaches is that deep networks can learn the distribution of possible programs from data and use that to guide the search.

Neural nets can also completely replace search-based synthesis by learning to generate program text given input requirements. Menon et al. [MTG*13] explored the use of data-driven methods for program induction. The idea has been extended in various ways to incorporate deep learning for program synthesis [DUB*17, PMS*17, RDF16]. For example, RobustFill [DUB*17] is an end-to-end neural program synthesizer designed for text editing. These approaches rely purely on neural nets, leading to a huge increase in speed, but at the expense of flexibility, generality, and completeness: while neural nets are good at capturing complex patterns in data, they face challenges when generalizing to novel, more complex scenarios. Such generalization can be critical for successful program synthesis itself, as well as for downstream tasks such as 3D shape and scene synthesis.

As we will see later in Section 7, the graphics and vision community have applied and extended neural program synthesis approaches for procedural shape and scene synthesis.

4.5. Summary

To summarize our discussion of different synthesis methods for structured shapes and scenes, we provide a flowchart detailed the circumstances under which each should be used (Figure 12). When only a few training examples are available, or the input provided to the system is some other form of user-specific constraint, constraint-based program synthesizers are the best fit (though they require some effort up-front in order to author the DSL in which output programs will be synthesized). For small datasets of examples, deep learning methods can't be used, so more "classic" probabilistic models are the best bet. If the type of content to be modeled has a fixed structure (e.g. a fixed grid structure, or some maximum number of parts which can only connect in so many ways), probabilistic graphical models such as Bayesian networks are applicable. If the structure can vary dynamically across the dataset, a PCFG or probabilistic program is a better fit (though learning algorithms are only known for tree-structured content). Lastly, if a large amount of training data is available, deep neural networks are the best choice. If the structures to be generated feature strong global coherence, then a latent variable model such as a VAE or GAN is desirable. For

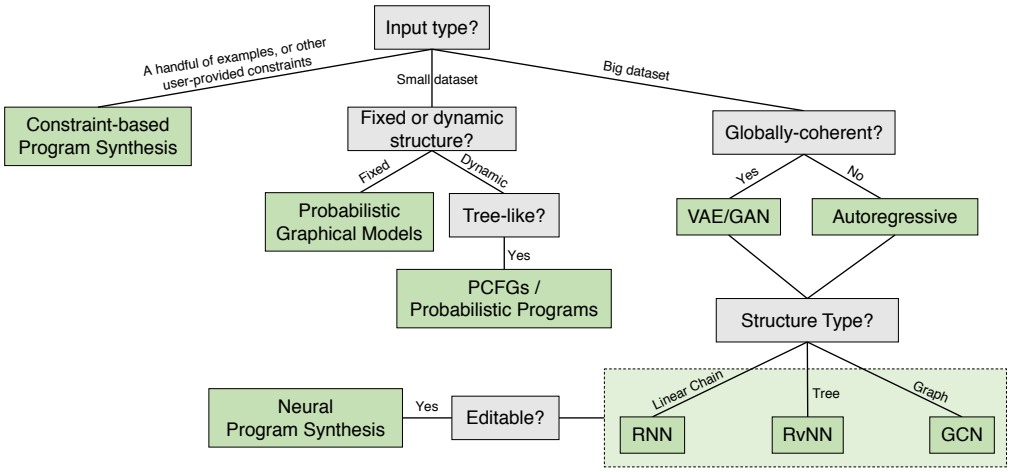


Figure 12: *A flowchart detailing the conditions to which each structure synthesis method discussed in Section 4 is best suited.*

more flexible, locally-coherent structures, an autoregressive model may suffice. Regardless, the particular neural network architecture used to generate the structure depends on the the type of structure: RNNs are the natural choice for linear chains, RvNNs are specialized for trees, and GCNs handle general graphs. Finally, if the application at hand demands a readable and editable representation of the output structure, neural program synthesis methods should be investigated.

5. Application: Part-based Shape Synthesis

Structural representations of individual 3D objects typically rely on compact encodings of discrete components of a shape. These components are commonly semantic or functional shape parts, e.g. the seat, back, legs and arms of a chair, or the head, torso, arms and legs of a human. In some cases, the parts may not possess obvious semantic labels, but still have meaningful cognitive identity and separation, e.g. primitive shapes defining a more complex object. In either case, part-based generative models typically rely on pre-segmented shape repositories, with or without part labels. While this adds a preprocessing overhead to be addressed by manual or semi/fully-automatic means [YKC*16, YSS*17], the major benefit is a model that can focus on the higher-level aspects of shape structure in terms of part layout and compatibility. The parts themselves in the final output can be repurposed from existing shapes, or synthesized separately in a factored representation. This leads to two main themes, which we now discuss.

Structure synthesis with repurposed parts. Several papers have developed systems that allow users to create new 3D shapes by assembling existing parts. Such systems provide automated aids for retrieving relevant parts, assembling them, and exchanging one part for another [FKS*04, KJS07, JTRS12]. Here, we will focus on methods that rely on learned generative models, instead of manual or heuristic-based approaches.

A variety of methods learn variations of a fixed template structure. While pioneering work on human modeling explored structured variations of body shapes [ASK*05], later work explored

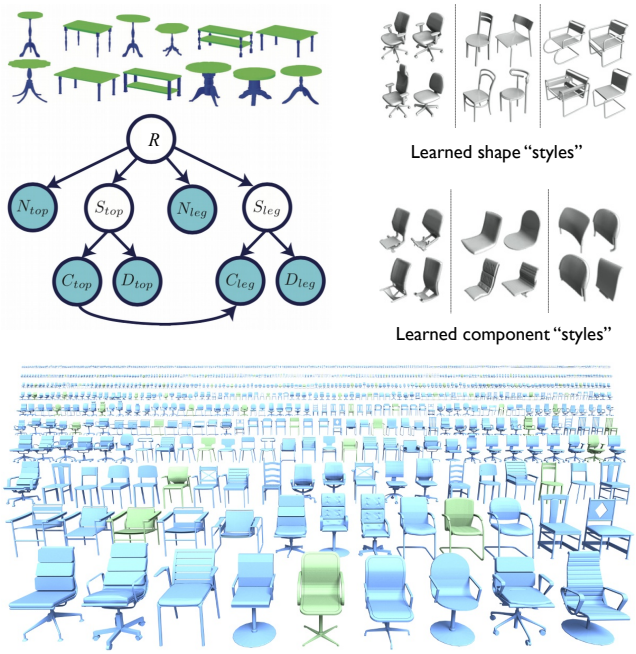


Figure 13: Top left: *A Bayesian network [KCKK12] trained on a toy table dataset with two part labels. R and S are latent variables that mediate correlations between observed features (blue circles).* Top right: *A full network trained on segmented and labeled chairs learns to associate stylistic clusters with different latent variable values.* Bottom: *Novel chairs (blue) sampled from a model trained on 88 chairs (green). Note that stylistic constraints are preserved.*

other domains as well, e.g. Fish et al. [FAvK*14] proposed a "meta-representation" that captured correlations between man-made shape parts and could be used to guide interactive editing.

Template models are limited by their fixed structure. To capture greater variation in structural topology, Chaudhuri et al. [CKGK11] and Kalogerakis et al. [KCKK12] trained Bayesian networks on

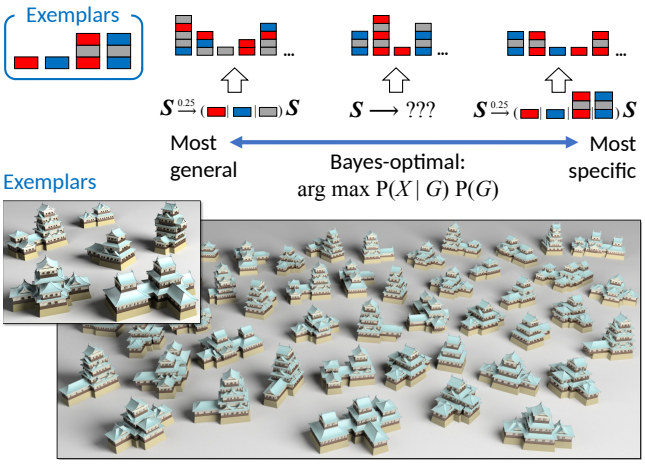


Figure 14: Top: *Given hierarchical exemplars constructed from a small set of building blocks, Bayesian model merging [TYK*12] induces a probabilistic grammar that captures the training set as high-probability derivations, while being compact.* Bottom: *Novel buildings sampled from a grammar induced from a small set of exemplars.*

segmented and labeled 3D repositories, modeling probability distributions over features, cardinalities, and adjacencies of labeled parts. While the former method was designed for suggesting parts that could augment an incomplete shape during interactive modeling, the latter was suitable for unsupervised shape synthesis by probabilistic sampling. Notable aspects of the Bayesian network of Kalogerakis et al. are latent variables that implicitly learned geometric clusters roughly corresponding to shape and part "styles" (Figure 13). In both cases, the sampled part features are used to retrieve parts from the training repository that are procedurally assembled into the final shape. Bayesian networks were also employed by Fan and Wonka [FW16] to synthesize residential buildings: their method first samples statistical attributes of the building from the graphical model, and then uses an optimization step that synthesizes an actual 3D shape obeying the attributes by a combination of procedural generation and part retrieval. Jaiswal et al. [JHR16] developed a factor graph that is used to infer part suggestions from a segmented dataset, *without* the need for part labels.

Further variations are captured by probabilistic grammars (PCFGs), which also conveniently produce organizational hierarchies for sampled shapes. While shape synthesis with grammars has a rich history [PL96, MWH*06], there are very few works that *learn* the grammar from examples. Bokeloh et al. [BWS10] reconstruct a grammar from a single exemplar shape, utilizing symmetries in how parts are connected to each other. Talton et al. [TYK*12] use Bayesian model merging to induce a grammar from several exemplar shapes assembled from a small library of parts (Figure 14). The method requires known hierarchies for all exemplars, and constructs the "most specific grammar" describing the collection as the union of these hierarchies. It then repeatedly merges (and occasionally splits) rules to obtain a more compact grammar that can still produce all the hierarchies with high probability, thus optimizing a Bayesian objective. Rule probabilities are

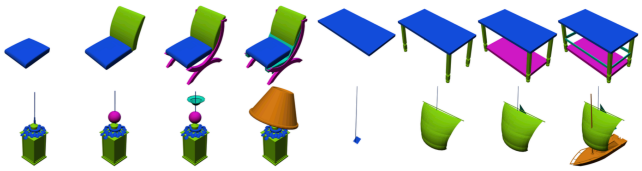


Figure 15: *The ComplementMe neural networks [SSK*17] learn to incrementally construct a shape, one part at a time.*

estimated via expectation-maximization. Martinović et al. [MG13] developed a similar method for building facade generation.

Taking generality still further, Ritchie et al. [RJT18] presented a method to induce simple probabilistic programs that describe a set of exemplar shapes constructed from a small library of parts. A preprocessing phase organizes the parts in each shape into a consistent hierarchy, followed by a learning phase that assembles the program from predefined synthesis and transformation building blocks. While the programs studied in this work are essentially extended grammars, they lay the foundation for induction of more complicated 3D programs.

In a different direction, Sung et al. [SSK*17] developed ComplementMe, a system that trains a deep neural network to retrieve parts that can augment an incomplete shape. The application is similar to [CKGK11] and [JHR16], but the approach avoids both graphical models and the need to have a labeled dataset. A second network predicts part placements. Together, the system is powerful enough to incrementally generate plausible complete shapes without human input (Figure 15).

Synthesis of both structure and parts. Instead of assembling existing parts from a repository, it is also possible to synthesize the parts themselves, along with their organizing structure. A (relatively) early learned generative model that tackled this problem was by Huang et al. [HKM15]. They developed a Deep Boltzmann Machine to build part templates for groups of structurally similar shapes in a collection, and establish part and point correspondences between the shapes via a learned part-aware deformation model that smoothly maps between surface points on different shapes. The grouping, and a labeled segmentation for a shape in each group, are input to the training. The model is generative and can be used to synthesize shapes as labeled point clouds with template deformations: for visual quality, the final output replaces the points of each part with a deformed existing mesh part (Figure 16). A recent paper by Gao et al. [GYW*19] adopts a similar "deform-and-assemble" approach: a two-level VAE combines structure prediction with part synthesis, where each part is modeled as a deformed cuboid. The model jointly predicts the layout of parts as well as the local details of each part. Instead of learning deformations, Li et al. [LNX19] train a VAE-GAN to predict voxelized parts (one for each of a fixed set of semantic labels) from scratch. A second network module learns assembly transformations to produce a coherent shape from the synthesized parts.

A different approach to shape generation is *structure-aware refinement*. For instance, Balashova et al. [BSW*18] propose a voxel VAE that is augmented by a structure-aware loss. A network module tries to map a pre-specified, deformable keypoint arrangement

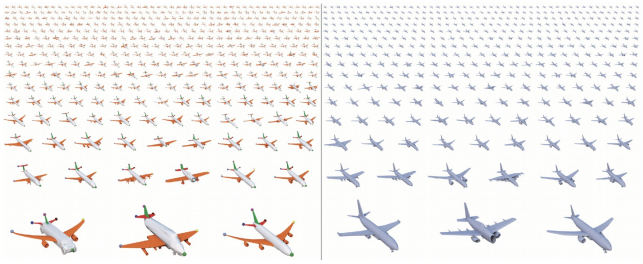


Figure 16: *A Deep Boltzmann Machine [HKM15] can infer semantic parts and point correspondences in a shape collection (left), while producing a structure-aware generative model which can create novel shapes (right).*

to a synthesized voxelized shape. The predicted arrangement respects a structural prior trained on shapes annotated with keypoints. The voxel grid is refined with a structure consistency loss that tries to ensure the presence of occupied voxels near each predicted keypoint. More explicit part information is introduced by Wang et al. [WSH*18], which trains a GAN to output a voxelized shape with per-voxel part labels. The GAN uses both global (shape-level) and local (part-level) adversarial losses to enhance shape plausibility. The GAN output is fed to a denoising autoencoder that refines and re-assembles the individual parts.

The above methods rely on shapes having a fixed structural topology, or a set of topologies represented as deformable templates. Hence, they typically require all parts to conform to a small set of labels. However, shapes in the real world can have (a) extensive structural variation, (b) difficult-to-label components, and (c) fine-grained details that themselves have compositional structure. To address these challenges, Li et al. [LXC*17] propose GRASS, a recursive VAE that models layouts of arbitrary numbers of unlabeled parts, with relationships between them. Based on the observation that shape parts can be hierarchically organized [WXL*11], the method utilizes an RvNN for encoding and decoding shape structures. The hierarchical organization follows two common perceptual/cognitive criteria for recursive merging: a mergeable subset of parts is either an *adjacent* pair, or part of a *symmetry* group. Three different symmetries are each represented by a generator part plus further parameters: (1) pairwise reflectional symmetry; (2) k-fold rotational symmetry; and (3) k-fold translational symmetry. Separate adjacency and symmetry encoders/decoders handle the two criteria, and a classifier is trained in parallel to predict which should be applied during decoding.

Training data is created by randomly sampling multiple hierarchies that respect the merging criteria [WXL*11] for each segmented (but not labeled) training shape. Despite the absence of consistent, ground truth training hierarchies, the low-capacity RvNN modules converge to predict relatively consistent hierarchies for any input shape. Figure 17 shows an illustration of the symmetry hierarchy of a chair model and its RvNN encoder. Since the encoder and decoder are linked by a generative VAE, the method can sample feature vectors from the latent space and decode them to synthesize novel shapes, complete with part hierarchies. A GAN loss improves the plausibility of the output, and a separate network generates fine-grained part geometry for the synthesized layout. Figure 17 also

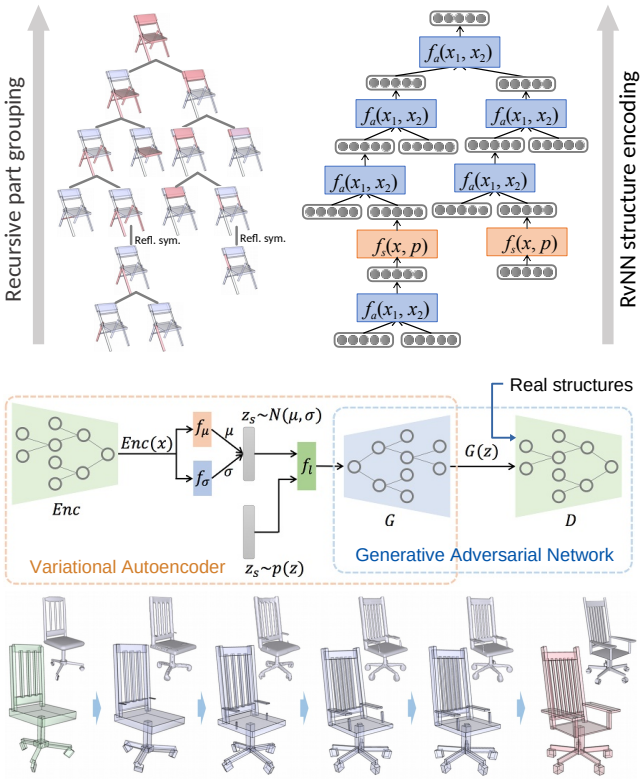


Figure 17: Top: *The symmetry hierarchy of a chair model and the corresponding RvNN encoder. Structure decoding reverses the process.* Middle: *The recursive VAE-GAN architecture of GRASS [LXC*17].* Bottom: *An example of topology-varying shape interpolation with GRASS. Note the changes in spoke, slat and arm counts.*

shows the RvNN pipeline, and topology-varying interpolation via the latent space.

Recent work by Zhu et al. [ZXC*18] uses an RvNN to iteratively refine a rough part assembly into a plausible shape, optionally adding or removing parts as needed. The method learns a *substructure prior* to handle novel structures that only locally resemble training shapes. Mo et al. [MGY*19a] extended the RvNN approach with graph neural networks as the encoder/decoder modules, instead of traditional single- or multi-layer perceptrons as in GRASS. This allows direct encoding of n-ary adjacencies, leading to better reconstruction and more cognitively plausible hierarchies. In very recent work, the same group proposed structure-aware shape editing by encoding and decoding shape differences (deltas) using the hierarchical graph representation. [MGY*19b]. Dubrovina et al. [DXA*19] also proposed structure-aware editing, by factorizing the latent space of an autoencoder into independent subspaces for different part types. However, their network is not strictly speaking a *generative* model.

Structure-aware models as priors for cross-modal synthesis. The learned models described above can not only be sampled directly for unconstrained shape synthesis, they can also be used

to generate shapes from cues in other modalities. For instance, linguistic predicates can be used to interactively create new 3D shapes guided by discrete [CKGF13] or continuous [YCHK15] generative priors. Similarly, structure-aware priors have been used to reconstruct 3D shapes from incomplete, noisy and partially occluded scans [SKAG15, ZXC*18], as well as from 2D RGB images [NLX18]. A full discussion of existing and anticipated work in this space is beyond the scope of the current report.

6. Application: Indoor Scene Synthesis

With the resurgence of computer surveillance, robotics, virtual and augmented reality (VR/AR), as well as home safety and intelligence applications, there has been an increasing demand for virtual models of indoor scenes. At the same time, the recent rapid developments of machine learning techniques have motivated modern approaches to scene analysis and synthesis to adopt various data-driven and learning-based paradigms. In this section, we survey notable works on indoor scene synthesis with an emphasis on learning and generating scene structures. We direct the reader to [Ma17] for a more comprehensive, albeit less up-to-date, coverage on this topic, as well as scene analysis and understanding.

There are different ways to examine or classify scene synthesis methods. In terms of input, earlier works on probabilistic models, e.g., [FRS*12], generates a new scene by taking a random sample from a learned distribution, while recent works on deep generative neural networks, e.g., [LPX*19], can produce a novel scene from a random noise vector. The input can also be a hand sketch [XCF*13], a photograph [ISS17, LZW*15], natural language commands [MGPF*18], or human actions/activities [FLS*15, MLZ*16]. In terms of output, while most methods have been designed to generate room layouts with 3D furniture objects, some methods learn to produce floor or building plans [MSK10, WFT*19]. Our coverage will mainly organize the methods based on the methodologies applied, e.g., probabilistic models vs. deep neural networks, interactive vs. fully automated modeling, and wholistic approaches vs. progressive synthesis.

Shape parts vs. scene objects. The basic building blocks of an indoor scene are 3D objects whose semantic and functional composition and spatial arrangements follow learnable patterns, but still exhibit rich variations in object geometry, style, arrangements, and other relations, even within the same scene category (e.g., think of all the possible layouts of kitchens, living rooms, and bedrooms). It is natural to regard parts in a 3D object in the same vein as objects in a 3D scene. However, while object-level part-part relations and scene-level object-object relations are both highly structured, there are several important distinctions between them which have dictated how the various scene generation methods are designed:

- As first observed by Wang et al. [WXL*11] and then reinforced by various follow-up works, the constituent parts of a 3D object are strongly governed by their connectivity and symmetry relations. In contrast, relations between objects in a scene are much *looser*. For example, many objects which are clearly related, e.g., sofas and sofa tables, desks and chairs, are rarely physically connected. Also, due to perturbations arising from human usage, symmetries (e.g., the rotational symmetry of dining chairs around a dining table) between objects are often not realized.
- In a 3D scene, closely related objects may not even be in close proximity. For example, TVs and sofas are often found together in living rooms as they are related by the human action "watching TV", but that very action dictates that TVs are almost never close to sofas geometrically. Indeed, the object-object relations in a scene are often more about *semantics* or *functionality* than geometric proximity. Unlike symmetry and connectivity, which are central to part-part relations in objects, functional and semantic relations are more difficult to model and learn.
- Alignment between objects is often a prerequisite to further analysis. Despite the rich geometric and structural variations that may exist within a category of man-made objects, e.g., chairs or airplanes, a good alignment is typically achievable. Such an alignment would bring out more predictability in the positioning of object parts, so as to facilitate subsequent learning tasks. On the other hand, it is significantly more difficult, or even impossible, to find reasonable alignments between scene layouts, even if they are all of kitchens, bedrooms, or living rooms.

Scene retrieve-n-adjust. A straightforward way to generate new scenes when given a large dataset of available scenes is to *retrieve* "pieces" of sub-scenes from the dataset, based on some search criteria, and then adjust the retrieved pieces when composing them into the final scene. While there may not be a formal training process in these approaches, some rules/criteria often need to be learned from the data to guide the scene adjustment step.

In Sketch2Scene, Xu et al. [XCF*13] propose a framework that turns a 2D hand drawing of a scene into a 3D scene composed of semantically valid and well arranged objects. Their retrieval-based approach centers around the notion of *structure groups* which are extracted from an existing 3D scene database; this can be regarded as a learning process. A structure group is a compact summarization of re-occurring relationships among objects in a large database of 3D indoor scenes. Given an input sketch, their method performs co-retrieval and co-placement of relevant 3D models based on the "learned" structural groups to produce a 3D scene.

Given a single photograph depicting an indoor scene, Liu et al. [LZW*15] reconstruct a corresponding 3D scene by first segmenting the photo into objects and estimating their 3D locations. This is followed by 3D model retrieval from an existing database by matching the line drawings extracted from the 2D objects and line features of the 3D models in the database.

Ma et al. [MGPF*18] use natural language commands as a prior to guide a classical retrieve-n-adjust paradigm for 3D scene generation. Natural language commands from the user are first parsed and then transformed into a semantic scene graph that is used to retrieve corresponding sub-scenes from an existing scene database. Known scene layouts from the database also allow the method to augment retrieved sub-scenes with other objects that may be implied by the scene context. For example, when the input command only speaks of a TV being in the living room, a TV stand is naturally inserted to support the TV object. The 3D scene is generated progressively by processing the input commands one by one. At each step, a 3D scene is synthesized by "splicing" the retrieved sub-scene, with augmented objects, into the current scene.

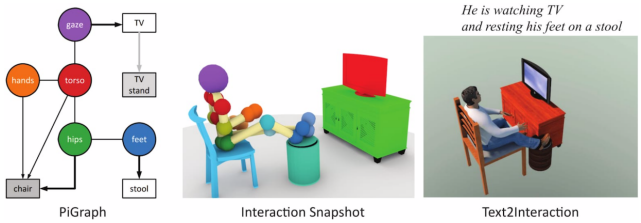


Figure 18: *A PiGraph [SCH*16] encodes relations, such as physical contacts, between human body parts and objects in an indoor scene. Such representations can be learned from captured human-scene interaction data and applied to generate interaction snapshots (middle) and enable text-driven interaction synthesis (right).*

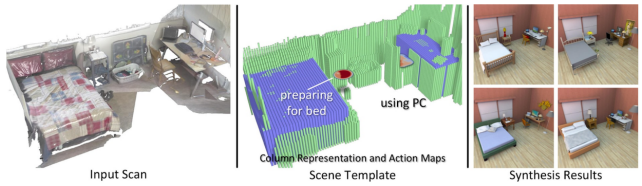


Figure 19: *Activity-centric scene synthesis [FLS*15] takes a RGBD scan of an indoor scene (left) and produces multiple well-modeled 3D scenes (right) which conform to the input scan and support plausible human activities therein. The synthesis is based on an estimated scene template which encodes where activities can occur and contains a coarse geometric representation describing the general layout of objects to support those activities (middle).*

Probabilistic synthesis. Example-based approaches to scene synthesis [MSK10, FRS*12, JLS12, MSSH14, ZHG*16, SCH*16] rely on probabilistic reasoning over exemplars from scene databases. Fisher et al. [FRS*12] develop Bayes networks and Gaussian Mixture Models (GMMs) to learn two probabilistic models for modeling sub-scenes (e.g. an office corner or a dining area): (a) object occurrence, which indicates what objects should be placed in the scene, and (b) layout optimization, which controls where the objects ought to be placed. Given an example scene, new variants can be synthesized based on the learned priors. For floor plan generation, Merrell et al. [MSK10] learn structured relations between features of architectural elements using a probabilistic graphical model. They also train a Bayesian network which represents a probability distribution over the space of training architectural programs. Once the Bayesian network is trained, a new floor plan can be generated by sampling from this distribution.

Progressive synthesis and interactive modeling. Instead of generating an entire scene or object arrangement in one shot, progressive synthesis inserts one object or sub-scene at a time, often based on user inputs such as language commands [MGPF*18] or human activities [MLZ*16, SCH*16]. As pointed out by Ma et al. [MLZ*16], in real life, indoor scenes evolve progressively as a result of human actions. From a technical point of view, the learning task for progressive synthesis is more localized and the synthesis process itself offers more controllability.

Sadeghipour et al. [KLTZ16] learn a probabilistic model from a large annotated RGB-D scene dataset and progressively sample the model to insert objects into an initial scene. The probabilistic co-occurrence model learned is represented as a *factor graph* which can encode pairwise object relations. What is new compared to Fisher et al. [FRS*12] is that the factor graph also encodes *higher-order* relations between objects, including support, symmetry, distinct orientations, and proximity, along with their probabilities.

For interactive modeling, Merrell et al. [MSL*11] present a suggestive modeling tool for furniture layout. Based on interior design guidelines, as well as room boundary constraints, the tool can generate furniture arrangements and present them to the user. The design guidelines are encoded as terms in a probability density function and the suggested layouts are generated by conditional sampling of this function. In Make it Home, Yu et al. [YYT*11] synthesize furniture layouts by optimizing a layout function which encodes spatial relationships between furniture objects. Another interactive modeling tool is ClutterPalette [YYT16] which allows a user to insert and position one object at a time to mess up an otherwise clean scene. As the user clicks on a region of the scene, the tool suggests a set of suitable objects to insert, which are based on support and co-occurrence relations learned from data.

Human action/activity-driven synthesis. We humans live in a 3D world and we constantly act on and interact with objects that surround us. There has been a great deal of work in robotics and computer vision on utilizing human-centric approaches for scene understanding. For scene *generation*, human actions or activities have also played a key role in some recent works [FLS*15, SCH*16, MLZ*16]. From a human-centric perspective, indoor scene modeling focuses more on whether the generated scenes are functional or plausible based on human usage scenarios.

Savva et al. [SCH*16] develop a probabilistic learning framework which connects human poses and object arrangements in indoor scenes, where the learning is based on real-world observations of human-scene interactions. They introduce a novel representation called Prototypical Interaction Graphs or PiGraphs (see Figure 18), which can capture representative (i.e., prototypical) physical contacts and visual attention linkages between 3D objects and human body parts. Joint probability distributions over human pose and object geometries are encoded in the PiGraphs and learned from data, and in turn, the learned PiGraphs serve to guide the generation of *interaction snapshots*, as shown in Figure 18. Such snapshots provide static depictions of human-object interactions.

In activity-centric scene synthesis, Fisher et al. [FLS*15] present a method which turns an input RGBD scan into multiple well-modeled 3D scenes. Rather than focusing on realistic 3D reconstruction, they produce scenes which would allow the same human activities as the captured environments; see Figure 19. Specifically, given the input scan of an indoor scene, their method estimates a scene template which captures both the prominent geometric properties of the scan as well as human activities that are likely to occur in the scene. To generate and place objects, activity models are trained using annotated scene datasets that are endowed with human activities. These activity models encode object distributions with respect to human activities and they would guide the scene synthesis based on the predicted activities in the template.

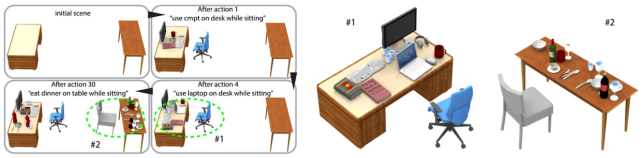


Figure 20: *Action-driven scene evolution [MLZ*16] progressively alters an indoor scene which started with just a desk and a dining table (left-top). Then a series of plausible human actions are selected and applied to insert objects into the scene that are likely associated with the actions. Three intermediate scene snapshots, after 1, 4, and 30 actions, are shown with zoom-ins for a better view (right).*

Ma et al. [MLZ*16] introduce *action-driven* 3D scene evolution, which is motivated by the observation that in real life, indoor environments evolve due to human actions or activities which affect object placements and movements; see Figure 20. The action model in their work defines one or more human poses, objects, and their spatial configurations that are associated with specific human actions, e.g., "eat dinner at table while sitting". Importantly, the action models are learned from annotated *photographs* in the Microsoft COCO dataset, not from 3D data which is relatively scarce, especially in terms of action labels and object/pose designations. Partial orders between the actions are analyzed to construct an action graph, whose nodes correspond to actions and edges encode transition probabilities between actions. Then, to synthesize an indoor scene, an action sequence is sampled from the action graph, where each sampled action would imply plausible object insertions and placements based on the learned action models.

Deep generative models. Highly structured models, including indoor scenes and many man-made objects, could be represented as a volume or using multi-view images and undergo conventional convolutionary operations. However, such operations are oblivious to the underlying structures in the data which often play an essential role in scene understanding. This could explain in part why deep convolutional neural networks, which have been so successful in processing natural images, had not been as widely adopted for the analysis or synthesis of 3D indoor scenes until recently.

Wang et al. [WSCR18] learn deep convolutional priors for indoor scene synthesis. Specifically, the deep convolutional neural network they develop generates top-down views of indoor scene layouts automatically and progressively, from an input consisting of a room architecture specification, i.e., walls, floor, and ceiling. Their method encodes scene composition and layout using multi-channel top-view depth images. An autoregressive neural network is then trained with these image channels to output object placement priors as a 2D distribution map. Scene synthesis is performed via a sequential placement of new objects, guided by the learned convolutional object placement priors. In one of the follow-ups, Ritchie et al. [RWaL19] develop a deep convolutional generative model which allows faster and more flexible scene synthesis.

More recently, Wang et al. [WLW*19] present PlanIT, a "plan-and-instantiate" framework for layout generation, which is applicable to indoor scene synthesis. Specifically, their method plans the structure of a scene by generating a relationship graph, and then instantiates a concrete 3D scene which conforms to that plan; see Figure 21. Technically, they develop a deep generative graph model based on message-passing graph convolution, and a specific model architecture for generating indoor scene relationship graphs. In addition, they devise a neurally-guided search procedure for instantiating scene relationship graphs, which utilize image-based scene synthesis models that are constrained by the input graph.

Like Wang et al. [WLW*19], Zhou et al. [ZWK19] also develop a graph neural network to model object-object relations in 3D indoor scenes and a message passing scheme for scene augmentation. Specifically, their network is trained to predict a probability distribution over object types that provide a contextual fit at a scene location. Rather than focusing on relations between objects in close proximity, the relation graphs learned by Zhou et al. [ZWK19] encodes both short- and long-range relations. Their message passing relies on an attention mechanism to focus on the most relevant surrounding scene context when predicting object types.

Li et al. [LPX*19] introduce GRAINS, a generative neural network which can generate plausible 3D indoor scenes in large quantities and varieties. Using their approach, a novel 3D scene can be generated from a random vector drawn from a Gaussian in a fraction of a second, following the pipeline shown in Figure 22. Their key observation is that indoor scene structures are inherently hierarchical. Hence, their network is not convolutional; it is a recursive neural network [SLNM11] (RvNN), which learns to generate hierarchical scene structures consisting of oriented bounding boxes for scene objects that are endowed with semantic labels.

Using a dataset of annotated scene hierarchies, they train a variational recursive autoencoder, or RvNN-VAE, which performs scene object grouping during its encoding phase and scene generation during decoding. Specifically, a set of encoders is recursively applied to group 3D objects in a scene, bottom up, and encodes information about the objects and their relations, where the resulting fixed-length codes roughly follow a Gaussian distribution. A new scene can then be generated top-down, i.e., hierarchically, by decoding from a randomly generated code.

Similar to GRAINS, Zhang et al. [ZYM*18] train a generative model which maps a normal distribution to the distribution of primary objects in indoor scenes. They introduce a 3D object arrangement representation that models the locations and orientations of objects, based on their size and shape attributes. Moreover, their scene representation is applicable for 3D objects with different multiplicities or repetition counts, selected from a database. Their generative network is trained using a hybrid representation, by combining discriminator losses for both a 3D object arrangement representation and a 2D image-based representation.

Most recently, Wu et al. [WFT*19] develop a data-driven technique for automated floor plan generation when given the boundary of a residential building. A two-stage approach is taken to imitate the human design process: room locations are estimated first and then walls are added while conforming to the input building boundary. They follow a heuristic generative process which begins with the living room and then continues to generate other rooms progressively. Specifically, an encoder-decoder network is trained to generate walls first, where the results can be rough; this is followed

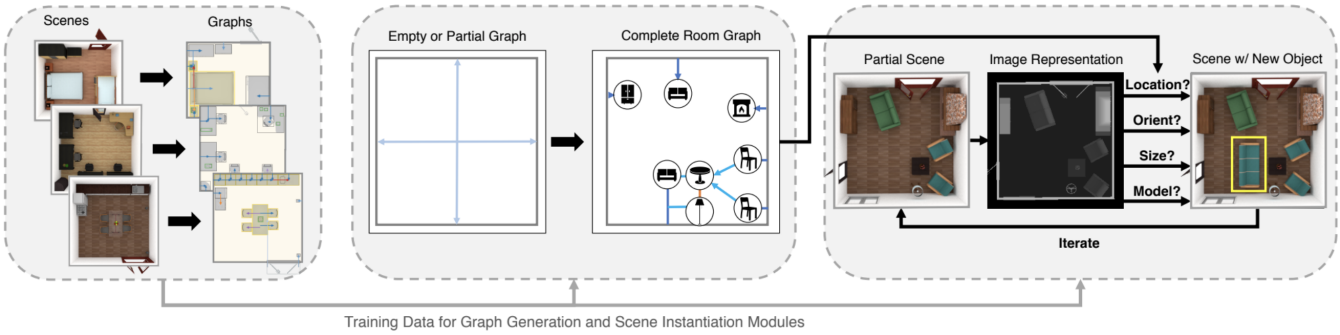


Figure 21: *Scene synthesis pipeline of PlanIT [WLW*19]. Relation graphs are automatically extracted from scenes (left), which are used to train a deep generative model of such graphs (middle). In the figure, the graph nodes are labeled with an icon indicating their object category. Image-based reasoning is then employed to instantiate a graph into a concrete scene by iterative insertion of 3D models for each node (right).*

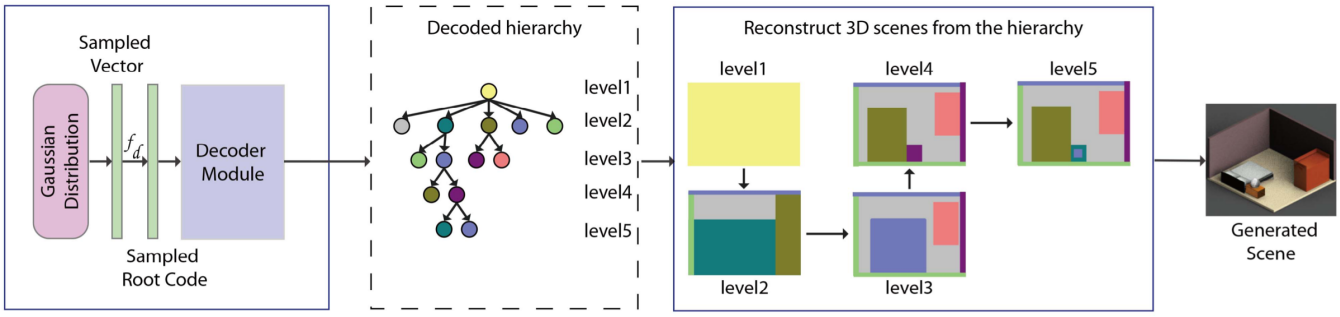


Figure 22: *Overall pipeline of scene generation using GRAINS [LPX*19]. The decoder of the trained RvNN-VAE (variational autoencoder) turns a randomly sampled code from the learned distribution into a plausible indoor scene hierarchy composed of oriented bounding boxes (OBBs) with semantic labels. The labeled OBBs are used to retrieve 3D objects to form the final 3D scene.*

by refining the generated walls to vector representations using dedicated rules. A key contribution of their work is RPLAN, a large-scale, densely annotated dataset of floor plans from real residential buildings, which is used to train their neural network.

7. Application: Visual Program Induction

In addition to synthesizing novel 3D content, structure-aware generative models can also be used to infer the underlying structure of existing 3D content (to facilitate editing and other downstream processing tasks). Here, the visual input can be 3D shapes and scenes (voxels, point clouds, etc.), or even 2D images. In this section, we focus on one particularly active research area along this theme: *visual program induction*, i.e. synthesizing a plausible program that recreates an existing piece of 3D content. While there are non-learning-based instances of this paradigm [DIP*18, NAW*19], we focus on learning-based methods in this section.

One line of work attempts to recover shape-generating programs from an existing 3D shape. The earliest instances of this paradigm are works which learn to reconstruct 3D shapes via a set or sequence of simple geometric primitives, which can be interpreted as very simple shape-generating programs [TSG*17, ZYY*17]. As shown in Figure 23, the model from Tulsiani et al. [TSG*17] learns to decompose shapes into primitives and uses Chamfer distance, which computes the difference between the reconstructed shape and the input shape, as the loss function during training. A more complex class of program is constructive solid geometry (CSG); Sharma et al. [SGL*18] presented an autoregressive generative model which synthesizes CSG programs that represent a given input shape (in both 2D and 3D domains).

Very recently, Tian et al. [TLS*19] presented an autoregressive model that takes an input 3D shape (represented as a volumetric grid) and outputs an imperative 3D shape program consisting of loops and other high-level structures (Figure 24). Their system also includes a neural network for *executing* these programs in a differentiable manner, which allows the generative model to be trained in a self-supervised manner (i.e. generate programs, execute programs, compare output 3D shape to input 3D shape). An overview of their model can be found in Figure 25. The model was later extended by Ellis et al. [ENP*19], where they formulate the problem as policy learning in reinforcement learning, and use deep networks to suggest the action (i.e., what the next program block is) directly.

Another line of work aims to perform visual program induction directly from 2D images. Kulkarni et al. [KKTM15] formulated this "inverse graphics" problem as inference in probabilistic programs that produce 2D contours or images given an underlying 3D

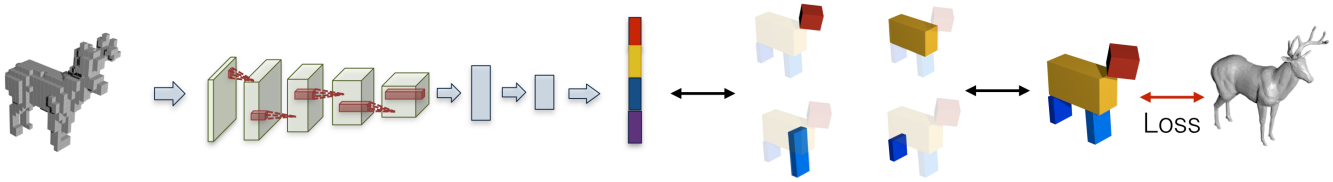


Figure 23: *Overall pipeline of reconstructing 3D shapes via a set of geometric primitives [TSG*17]. The model learns to decompose shapes into primitives and uses Chamfer distance, which computes the difference between the reconstructed shape with the input shape, as the loss function to train the inference network.*

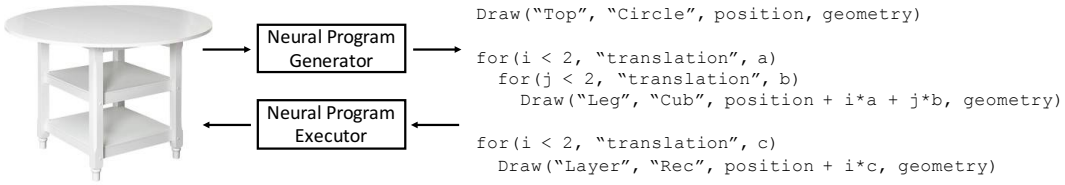


Figure 24: *The* shape program *representation aims to explain an input shape with high-level programs that encodes basic parts, as well as their structure such as repetition [TLS*19].*

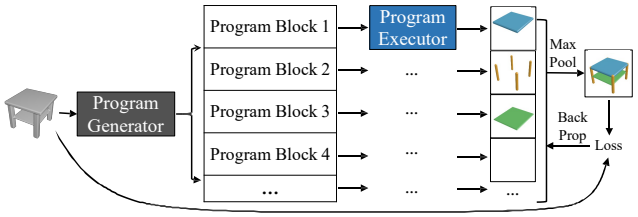


Figure 25: *The model for inferring and executing 3D shape programs [TLS*19]. At each step, it infers the next program block (e.g. a set of cuboids as legs), executes the program block, and compares the current shape with the input to suggest what to synthesize next.*

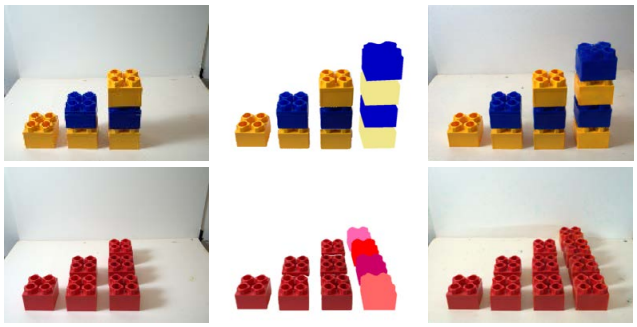


Figure 26: *Visual program induction allows efficient and flexible scene manipulation, such as layout extrapolation [LWR*19].*

model. Ellis et al. [ERSLT18] combined autoregressive generative models with constraint-based program synthesis to infer programs which generate 2D diagrams directly from hand-drawn sketches. While these approaches focus on cases where the input describes a single shape, there have also been attempts to apply visual program induction to multi-object 3D scenes, and to use the inferred programs for downstream tasks such as image editing [LWR*19]. As shown in Figure 26, visual program induction allows efficient and flexible scene manipulation such as layout extrapolation.

8. Conclusions and Future Work

In this report, we have surveyed the history of and state-of-the-art methods for learning structure-aware generative models of 3D shapes and scenes. We have discussed the array of possible representations of both individual structural atoms (parts of objects; objects in scenes) as well as the structural patterns which combine them together. We have explored how these different representations admit different generative modeling methods, each with unique strengths and weaknesses. Finally, we gave an overview of applications of these technologies to different 3D shape and scene generation tasks.

While the field of structure-aware generative modeling has already delivered impressive results, there is still much to be done. To conclude this report, we offer thoughts on several of the most important directions for improvement and further research:

Improving deep generative models of structure. It is very challenging to learn to generate the kinds of structure representations described in Section 3.2. These are complex data structures with discrete topological connections as well as continuous attributes. Thus, one way to improve structure-aware 3D generative models is to develop better, more reliable generative models of structures (e.g. trees, graphs, hierarchical graphs, programs). This is, to some extent, a task for machine learning researchers. However, graphics researchers can potentially make contributions by leveraging domain-specific insights (e.g. building a generative model that is

specialized for certain graph topologies which commonly occur in 3D data).

Reducing dependence on structural supervision. To date, successes in learning-based 3D generative modeling have been due to supervised learning: training on large collections of 3D shapes and scenes annotated with 'ground-truth' structure. For example, the StructureNet architecture [MGY*19a] was trained on the PartNet dataset [MZC*19], which is a subset of ShapeNet in which each object is decomposed into a hierarchical graph of parts and every part is labeled. These kinds of structural annotations are costly to produce; heavy reliance on them limits the domains in which these methods can be applied. It would be better if structure-aware generative models could be learned directly from raw 3D content (e.g. meshes or point clouds) without having to manually annotate the structure of each training item.

One potential way to do this is self-supervised learning (sometimes also called predictive learning). The idea is to use a training procedure in which the model generates a structure-aware representation, converts that representation into a raw, unstructured representation, and then compares this to the ground-truth unstructured geometry. Training to minimize this comparison loss allows the model to learn without ever having access to the ground-truth structure; instead, it is forced to *discover* a structure which explains the geometry well.

Self-supervision has been applied successfully to train models which can reconstruct a 3D shape from a 2D image without having access to ground truth 3D shapes at training time [YYY*16]. As far as we are aware, the only instance of self-supervised learning for structure-aware generative models is the work of Tian et al. [TLS*19], which still required supervised pre-training. The challenge with applying self-supervised learning to structure-aware generative models is that the entire training pipeline needs to be end-to-end differentiable in order to train via gradient-based methods, but generating structures inherently requires non-differentiable, discrete decisions. Sometimes it is possible to learn a smooth, differentiable approximation to this discrete process (this is the method employed by Tian et al.). It is also possible that advances in reinforcement learning will allow self-supervised structure generation with non-differentiable training pipelines.

Improving the coupling of structure and geometry. As mentioned in Section 3, there are two parts to a representation: the geometry of individual structural atoms and the structure that combines them. We surveyed the different possible representations for each and different generative models for them. But we have treated them as separate components, when they are in actuality coupled: structure influences the geometry and vice-versa. Most prior work largely ignores this coupling and trains separate generative models for the structure vs. the fine-grained geometry of individual atoms. One recent work recognizes the necessity of this coupling and defines a structure-aware generative model that attempts to better couple the two [WWL*19]. Another line of research tries to impose structural constraints (symmetries) on implicit surface representations through learning a structured set of local deep implicit functions [GCV*19, GCS*19]. Such structured implicit functions ensure both accurate surface reconstruction and structural correctness. Overall, this direction of research is still under-explored.

Modeling the coupling of geometry and appearance. The generative models we have surveyed generate geometric models without materials or textures, but these are critical components of actual objects and scenes, and they are strongly coupled to the structure and geometry (i.e. structure and geometry provide strong cues to material and texture appearance). There is some work on inferring materials for parts of objects [Arj12, LAK*18] and for objects in scenes [CXY*15], as well as more recent explorations on generating colored and textured 3D objects [CCS*18,ZZZ*18]. In general, however, this area is also under-explored.

Modeling object and scene style. Material and texture appearance contributes strongly to an object or scene's perceived "style." State-of-the-art generative models of 2D images have developed powerful facilities for controlling the style of their synthesized output [KLA19]; in comparison, 3D generative models lag behind. Style can be a function of both geometry (e.g. a sleek modern chair vs. ornate antique chair) as well as structure (e.g. a compact, cozy living space vs. an airy, open one).

One form of style that is worth mentioning especially is "messiness." Computer graphics has historically suffered from producing imagery that looks 'too perfect' and thus appears obviously fake. In the realm of artist-created 3D content, this problem has largely been solved via more photorealistic rendering algorithms and higher-detail modeling and texturing tools. However, it is still a problem with content created by learned generative models. It is particularly problematic for 3D scenes: generating scenes which appear 'cluttered' in a naturalistic way is extremely difficult and still an open problem.

Modeling object and scene functionality. Existing generative models focus on visual plausibility: that is, they attempt to generate 3D content that *looks* convincingly like a particular type of entity (e.g. a chair). However, the content being generated by these models increasingly needs to do more than just look good: it needs to be interacted with in a virtual environment, or even physically fabricated and brought into the real world. Thus, we need generative models that create *functionally* plausible 3D content, too. Functional plausibility can have a variety of meanings, ranging from ensuring that generated objects are equipped with expected affordances (e.g. the drawers of a generated desk can open), that generated scenes support desired activities (e.g. that the living room of a five-bedroom house contains enough seating arranged appropriately to allow the whole family to watch television), and that synthesized content is physically realizable (i.e. it will not collapse under gravity or under typical external loads placed upon it).

9. Author Bios

Siddhartha Chaudhuri is a Senior Research Scientist in the Creative Intelligence Lab at Adobe Research, and Assistant Professor (on leave) of Computer Science and Engineering at IIT Bombay. He obtained his Ph.D. from Stanford University, and his undergraduate degree from IIT Kanpur. He subsequently did postdoctoral research at Stanford and Princeton, and taught for a year at

Cornell. Siddhartha's work combines geometric analysis, machine learning, and UI innovation to make sophisticated 3D geometric modeling accessible even to non-expert users. He also studies foundational problems in geometry processing (retrieval, segmentation, correspondences) that arise from this pursuit. His research themes include probabilistic assembly-based modeling, semantic attributes for design, generative neural networks for 3D structures, and other applications of deep learning to 3D geometry processing. He is the original author of the commercial 3D modeling tool Adobe Fuse, and has taught tutorials on data-driven 3D design (SIGGRAPH Asia 2014), shape "semantics" (ICVGIP 2016), and structure-aware 3D generation (Eurographics 2019).

Daniel Ritchie is an Assistant Professor of Computer Science at Brown University. He received his PhD from Stanford University, advised by Pat Hanrahan and Noah Goodman. His research sits at the intersection of computer graphics and artificial intelligence, where he is particularly interested in data-driven methods for designing, synthesizing, and manipulating visual content. In the area of generative models for structured 3D content, he co-authored the first data-driven method for synthesizing 3D scenes, as well as the first method applying deep learning to scene synthesis. He has also worked extensively on applying techniques from probabilistic programming to procedural modeling problems, including to learning procedural modeling programs from examples. In related work, he has developed systems for inferring generative graphics programs from unstructured visual inputs such as hand-drawn sketches

Jiajun Wu is an Acting Assistant Professor at the Department of Computer Science, Stanford University, and will join the department as an Assistant Professor in Summer 2020. He received his Ph.D. and S.M. in Electrical Engineering and Computer Science, both from Massachusetts Institute of Technology, and his B.Eng. from Tsinghua University. His research interests lie in the intersection of computer vision (in particular 3D vision), computer graphics, machine learning, and computational cognitive science. On topics related to this STAR, he has worked extensively on generating 3D shapes using modern deep learning methods such as generative adversarial nets. He has also developed novel neural program synthesis algorithms for explaining structures within 3D shapes and scenes and applied them in downstream tasks such as shape and scene editing.

Kai (Kevin) Xu is a Professor at the School of Computer Science, National University of Defense Technology, where he received his Ph.D. in 2011. He conducted visiting research at Simon Fraser University (2008-2010) and Princeton University (2017-2018). His research interests include geometry processing and geometric modeling, especially on data-driven approaches to the problems in those directions, as well as 3D vision and its robotic applications. He has published 70+ research papers, including 20+ SIGGRAPH/TOG papers. He organized two SIGGRAPH Asia courses and a Eurographics STAR tutorial, both on data-driven shape analysis and processing. He is currently serving on the editorial board of ACM Transactions on Graphics, Computer Graphics Forum, Computers & Graphics, and The Visual Computer. Kai has made several major contributions to structure-aware 3D shape analysis and modeling with data-driven approach, and recently with deep learning methods.

Hao (Richard) Zhang is a professor in the School of Computing Science at Simon Fraser University, Canada. He obtained his Ph.D. from the Dynamic Graphics Project (DGP), University of Toronto, and M.Math. and B.Math degrees from the University of Waterloo, all in computer science. Richard's research is in computer graphics with special interests in geometric modeling, analysis and synthesis of 3D contents (e.g., shapes and indoor scenes), machine learning (e.g., generative models for 3D shapes), as well as computational design, fabrication, and creativity. He has published more than 120 papers on these topics. Most relevant to the STAR topic, Richard was one of the co-authors of the first Eurographics STAR on structure-aware shape processing and taught SIGGRAPH courses on the topic. With his collaborators, he has made original and impactful contributions to structural analysis and synthesis of 3D shapes and environments including co-analysis, hierarchical modeling, semi-supervised learning, topology-varying shape correspondence and modeling, and deep generative models.

Acknowledgement

Kai Xu is supported in part by supported in part by NSFC (61572507, 61532003, 61622212) and Natural Science Foundation of Hunan Province for Distinguished Young Scientists (2017JJ1002).

References

[ADMG18] ACHLIOPTAS P., DIAMANTI O., MITLIAGKAS I., GUIBAS L.: Learning Representations and Generative Models for 3D Point Clouds. In *International Conference on Machine Learning (ICML)* (2018). 3

[Arj12] ARJUN JAIN AND THORSTEN THORMÄHLEN AND TOBIAS RITSCHEL AND HANS-PETER SEIDEL: Material Memex: Automatic material suggestions for 3D objects. In *Annual Conference on Computer Graphics and Interactive Techniques Asia (SIGGRAPH Asia)* (2012). 19

[ASK*05] ANGUELOV D., SRINIVASAN P., KOLLER D., THRUN S., RODGERS J., DAVIS J.: SCAPE: Shape Completion and Animation of People. *ACM Transactions on Graphics (TOG) 24*, 3 (2005), 408–416. 11

[BGB*17] BALOG M., GAUNT A. L., BROCKSCHMIDT M., NOWOZIN S., TARLOW D.: DeepCoder: Learning to Write Programs. In *International Conference on Learning Representations (ICLR)* (2017). 10

[BLRW16] BROCK A., LIM T., RITCHIE J. M., WESTON N.: Generative and discriminative voxel modeling with convolutional neural networks. In *Advances in Neural Information Processing Systems Workshops (NeurIPS Workshop)* (2016). 3

[BSW*18] BALASHOVA E., SINGH V. K., WANG J., TEIXEIRA B., CHEN T., FUNKHOUSER T.: Structure-aware shape synthesis. In *International Conference on 3D Vision (3DV)* (2018). 12

[BWS10] BOKELOH M., WAND M., SEIDEL H.-P.: A connection between partial symmetry and inverse procedural modeling. *ACM Transactions on Graphics (TOG) 29*, 4 (2010). 12

[CCS*18] CHEN K., CHOY C. B., SAVVA M., CHANG A. X., FUNKHOUSER T., SAVARESE S.: Text2Shape: Generating shapes from natural language by learning joint embeddings. In *Asian Conference on Computer Vision (ACCV)* (2018). 19

[CFG*15] CHANG A. X., FUNKHOUSER T., GUIBAS L., HANRAHAN P., HUANG Q., LI Z., SAVARESE S., SAVVA M., SONG S., SU H., XIAO J., YI L., YU F.: ShapeNet: An Information-Rich 3D Model Repository. *arXiv:1512.03012* (2015). 3

[CKGF13] CHAUDHURI S., KALOGERAKIS E., GIGUERE S., FUNKHOUSER T.: AttribIt: Content creation with semantic attributes. In *ACM Symposium on User Interface Software and Technology (UIST)* (2013), pp. 193–202. 14

[CKGK11] CHAUDHURI S., KALOGERAKIS E., GUIBAS L., KOLTUN V.: Probabilistic Reasoning for Assembly-Based 3D Modeling. *ACM Transactions on Graphics (TOG) 30*, 4 (2011), 35. 11, 12

[CLS19] CHEN X., LIU C., SONG D.: Execution-Guided Neural Program Synthesis. In *International Conference on Learning Representations (ICLR)* (2019). 10

[CRXZ19] CHAUDHURI S., RITCHIE D., XU K., ZHANG H.: Learning Generative Models of 3D Structures. Eurographics Tutorial. https://3dstructgen.github.io/, 2019. 2

[CXY*15] CHEN K., XU K., YU Y., WANG T.-Y., HU S.-M.: Magic Decorator: Automatic Material Suggestion for Indoor Digital Scenes. In *Annual Conference on Computer Graphics and Interactive Techniques Asia (SIGGRAPH Asia)* (2015). 19

[CZ19] CHEN Z., ZHANG H.: Learning Implicit Fields for Generative Shape Modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 3

[DGF*19] DEPRELLE T., GROUEIX T., FISHER M., KIM V., RUSSELL B., AUBRY M.: Learning elementary structures for 3D shape generation and matching. In *Advances in Neural Information Processing Systems (NeurIPS)* (2019). 4

[DIP*18] DU T., INALA J. P., PU Y., SPIELBERG A., SCHULZ A., RUS D., SOLAR-LEZAMA A., MATUSIK W.: InverseCSG: Automatic conversion of 3D models to CSG trees. *ACM Transactions on Graphics (TOG) 37*, 6 (2018). 17

[DMB08] DE MOURA L., BJØRNER N.: Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems* (2008), Springer, pp. 337–340. 10

[DMI*15] DUVENAUD D. K., MACLAURIN D., IPARRAGUIRRE J., BOMBARELL R., HIRZEL T., ASPURU-GUZIK A., ADAMS R. P.: Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NeurIPS)* (2015). 9

[DN19] DAI A., NIESSNER M.: Scan2Mesh: From unstructured range scans to 3D meshes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 3

[DUB*17] DEVLIN J., UESATO J., BHUPATIRAJU S., SINGH R., MOHAMED A.-R., KOHLI P.: RobustFill: Neural Program Learning under Noisy I/o. In *International Conference on Machine Learning (ICML)* (2017). 10

[DXA*19] DUBROVINA A., XIA F., ACHLIOPTAS P., SHALAH M., GRVOSCOT R., GUIBAS L.: Composite shape modeling via latent space factorization. In *IEEE International Conference on Computer Vision (ICCV)* (2019). 13

[EMSM*18] ELLIS K., MORALES L., SABLÉ-MEYER M., SOLAR-LEZAMA A., TENENBAUM J. B.: Library Learning for Neurally-Guided Bayesian Program Induction. In *Advances in Neural Information Processing Systems (NeurIPS)* (2018). 10

[ENP*19] ELLIS K., NYE M., PU Y., SOSA F., TENENBAUM J., SOLAR-LEZAMA A.: Write, execute, assess: Program synthesis with a repl. In *Advances in Neural Information Processing Systems (NeurIPS)* (2019). 17

[ERSLT18] ELLIS K., RITCHIE D., SOLAR-LEZAMA A., TENENBAUM J.: Learning to Infer Graphics Programs from Hand-Drawn Images. In *Advances in Neural Information Processing Systems (NeurIPS)* (2018). 5, 10, 18

[FAvK*14] FISH N., AVERKIOU M., VAN KAICK O., SORKINE-HORNUNG O., COHEN-OR D., MITRA N. J.: Meta-representation of shape families. *ACM Transactions on Graphics (TOG) 33*, 4 (2014). 11

[FKS*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by example. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 652–663. 11

[FLS*15] FISHER M., LI Y., SAVVA M., HANRAHAN P., NIESSNER M.: Activity-centric scene synthesis for functional 3D scene modeling. *ACM Transactions on Graphics (TOG) 34*, 6 (2015), 212:1–10. 14, 15

[FRS*12] FISHER M., RITCHIE D., SAVVA M., FUNKHOUSER T., HANRAHAN P.: Example-based synthesis of 3D object arrangements. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 135:1–11. 1, 14, 15

[FSG17] FAN H., SU H., GUIBAS L. J.: A point set generation network for 3D object reconstruction from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). 3

[FSH11] FISHER M., SAVVA M., HANRAHAN P.: Characterizing structural relationships in scenes using graph kernels. *ACM Transactions on Graphics (TOG) 30*, 4 (2011), 34. 4

[FW16] FAN L., WONKA P.: A probabilistic model for exteriors of residential buildings. *ACM Transactions on Graphics (TOG) 35*, 5 (2016), 155:1–155:13. 12

[GCS*19] GENOVA K., COLE F., SUD A., SARNA A., FUNKHOUSER T.: Deep structured implicit functions. In *IEEE International Conference on Computer Vision (ICCV)* (2019). 19

[GCV*19] GENOVA K., COLE F., VLASIC D., SARNA A., FREEMAN W. T., FUNKHOUSER T.: Learning shape templates with structured implicit functions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 19

[GFK*18] GROUEIX T., FISHER M., KIM V. G., RUSSELL B. C., AUBRY M.: AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). 3, 4

[GFRG16] GIRDHAR R., FOUHEY D. F., RODRIGUEZ M., GUPTA A.: Learning a Predictable and Generative Vector Representation for Objects. In *European Conference on Computer Vision (ECCV)* (2016). 3

[GG19] GEORGIA GKIOXARI JITENDRA MALIK J. J.: Mesh R-CNN. In *IEEE International Conference on Computer Vision (ICCV)* (2019). 3

[GGH02] GU X., GORTLER S., HOPPE H.: Geometry images. In *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (2002). 3

[GMR*08] GOODMAN N. D., MANSINGHKA V. K., ROY D. M., BONAWITZ K., TENENBAUM J. B.: Church: a language for generative models. In *Conference on Uncertainty in Artificial Intelligence (UAI)* (2008). 7

[GPAM*14] GOODFELLOW I., POUGET-ABADIE J., MIRZA M., XU B., WARDE-FARLEY D., OZAIR S., COURVILLE A., BENGIO Y.: Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NeurIPS)* (2014). 3, 8

[GPS17] GULWANI S., POLOZOV A., SINGH R.: Program synthesis. *Foundations and Trends® in Programming Languages 4*, 1-2 (2017), 1–119. 10

[GS14] GOODMAN N. D., STUHLMÜLLER A.: The Design and Implementation of Probabilistic Programming Languages. http://dippl.org, 2014. Accessed: 2015-12-23. 7

[Gul11] GULWANI S.: Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL)* (2011). 10

[GYW*19] GAO L., YANG J., WU T., YUAN Y.-J., FU H., LAI Y.-K., ZHANG H. R.: SDM-NET: Deep generative network for structured deformable mesh. In *Annual Conference on Computer Graphics and Interactive Techniques Asia (SIGGRAPH Asia)* (2019). 12

[GZE19] GROVER A., ZWEIG A., ERMON S.: Graphite: Iterative generative modeling of graphs. In *International Conference on Machine Learning (ICML)* (2019). 4

[HBL15] HENAFF M., BRUNA J., LECUN Y.: Deep convolutional networks on graph-structured data. *arXiv:1506.05163* (2015). 9

[HHF*19] HANOCKA R., HERTZ A., FISH N., GIRYES R., FLEISHMAN S., COHEN-OR D.: MeshCNN: A network with an edge. *ACM Transactions on Graphics (TOG) 38*, 4 (2019), 90. 3, 9

[HKM15] HUANG H., KALOGERAKIS E., MARLIN B.: Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces. *Computer Graphics Forum (CGF) 34*, 5 (2015). 1, 12, 13

[HSG11] HWANG I., STUHLMÜLLER A., GOODMAN N. D.: Inducing Probabilistic Programs by Bayesian Program Merging. *arXiv:1110.5667* (2011). 7

[ISS17] IZADINIA H., SHAN Q., SEITZ S. M.: IM2CAD. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). 14

[JHR16] JAISWAL P., HUANG J., RAI R.: Assembly-based conceptual 3D modeling with unlabeled components using probabilistic factor graph. *Computer-Aided Design 74* (2016), 45–54. 12

[JLS12] JIANG Y., LIM M., SAXENA A.: Learning object arrangements in 3D scenes using human context. In *International Conference on Machine Learning (ICML)* (2012). 15

[JTRS12] JAIN A., THORMÄHLEN T., RITSCHEL T., SEIDEL H.-P.: Exploring shape variations by 3D-model decomposition and part-based recombination. *Computer Graphics Forum (CGF) 31*, 2 (2012). 11

[KCKK12] KALOGERAKIS E., CHAUDHURI S., KOLLER D., KOLTUN V.: A probabilistic model of component-based shape synthesis. *ACM Transactions on Graphics (TOG) 31*, 4 (2012). 11

[KF09] KOLLER D., FRIEDMAN N.: *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. 6

[KJS07] KRAEVOY V., JULIUS D., SHEFFER A.: Model composition from interchangeable components. In *Pacific Conference on Computer Graphics and Applications (Pacific Graphics)* (2007). 11

[KKTM15] KULKARNI T. D., KOHLI P., TENENBAUM J. B., MANSINGHKA V.: Picture: A Probabilistic Programming Language for Scene Perception. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015). 17

[KL17] KLOKOV R., LEMPITSKY V.: Escape from cells: Deep kd-networks for the recognition of 3D point cloud models. In *IEEE International Conference on Computer Vision (ICCV)* (2017). 3

[KLA19] KARRAS T., LAINE S., AILA T.: A style-based generator architecture for generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 19

[KLTZ16] KERMANI Z. S., LIAO Z., TAN P., ZHANG H.: Learning 3D scene synthesis from annotated RGB-D images. *Computer Graphics Forum (CGF) 35*, 5 (2016), 197–206. 15

[KMP*18] KALYAN A., MOHTA A., POLOZOV O., BATRA D., JAIN P., GULWANI S.: Neural-Guided Deductive Search for Real-Time Program Synthesis from Examples. In *International Conference on Learning Representations (ICLR)* (2018). 10

[KW14] KINGMA D. P., WELLING M.: Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR)* (2014). 3, 8

[KW17] KIPF T. N., WELLING M.: Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)* (2017). 9

[LAK*18] LIN H., AVERKIOU M., KALOGERAKIS E., KOVACS B., RANADE S., KIM V., CHAUDHURI S., BALA K.: Learning material-aware local descriptors for 3D shapes. In *International Conference on 3D Vision (3DV)* (2018). 19

[Lez08] LEZAMA A. S.: *Program synthesis by sketching*. PhD thesis, University of California Berkeley, 2008. 10

[LMTW19] LU S., MAO J., TENENBAUM J. B., WU J.: Neurally-Guided Structure Inference. In *International Conference on Machine Learning (ICML)* (2019). 10

[LNX19] LI J., NIU C., XU K.: Learning part generation and assembly for structure-aware shape synthesis. *arXiv:1906.06693* (2019). 4, 12

[LPX*19] LI M., PATIL A. G., XU K., CHAUDHURI S., KHAN O., SHAMIR A., TU C., CHEN B., COHEN-OR D., ZHANG H.: GRAINS: Generative recursive autoencoders for indoor scenes. *ACM Transactions on Graphics (TOG) 38*, 2 (2019), 12:1–12:16. 4, 14, 16, 17

[LVD*18] LI Y., VINYALS O., DYER C., PASCANU R., BATTAGLIA P. W.: Learning deep generative models of graphs. *arXiv:1803.03324* (2018). 4

[LWR*19] LIU Y., WU Z., RITCHIE D., FREEMAN W. T., TENENBAUM J. B., WU J.: Learning to Describe Scenes with Programs. In *International Conference on Learning Representations (ICLR)* (2019). 18

[LXC*17] LI J., XU K., CHAUDHURI S., YUMER E., ZHANG H., GUIBAS L.: GRASS: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 52. 1, 13

[LYF17] LIU J., YU F., FUNKHOUSER T.: Interactive 3D Modeling with a Generative Adversarial Network. In *International Conference on 3D Vision (3DV)* (2017). 3

[LZW*15] LIU Z., ZHANG Y., WU W., LIU K., SUN Z.: Model-driven indoor scenes modeling from a single image. In *Graphics Interface* (2015). 14

[Ma17] MA R.: Analysis and modeling of 3D indoor scenes. *arXiv:1706.09577* (2017). 14

[MBBV15] MASCI J., BOSCAINI D., BRONSTEIN M., VANDERGHEYNST P.: Geodesic convolutional neural networks on Riemannian manifolds. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshop)* (2015). 3

[MG13] MARTINOVIĆ A., GOOL L. V.: Bayesian grammar learning for inverse procedural modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2013). 12

[MGPF*18] MA R., GADI PATIL A., FISHER M., LI M., PIRK S., HUA B.-S., YEUNG S.-K., TONG X., GUIBAS L., ZHANG H.: Language-driven synthesis of 3D scenes from scene databases. *ACM Transactions on Graphics (TOG) 37*, 6 (2018), 212:1–212:16. 14, 15

[MGY*19a] MO K., GUERRERO P., YI L., SU H., WONKA P., MITRA N., GUIBAS L.: StructureNet: Hierarchical graph networks for 3D shape generation. In *Annual Conference on Computer Graphics and Interactive Techniques Asia (SIGGRAPH Asia)* (2019). 5, 13, 19

[MGY*19b] MO K., GUERRERO P., YI L., SU H., WONKA P., MITRA N., GUIBAS L. J.: StructEdit: Learning structural shape variations. *arXiv:1911.11098* (2019). 13

[MLZ*16] MA R., LI H., ZOU C., LIAO Z., TONG X., ZHANG H.: Action-driven 3D indoor scene evolution. *ACM Transactions on Graphics (TOG) 35*, 6 (2016). 14, 15, 16

[MON*19] MESCHEDER L., OECHSLE M., NIEMEYER M., NOWOZIN S., GEIGER A.: Occupancy Networks: Learning 3D Reconstruction in Function Space. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 3

[MPJ*19] MICHALKIEWICZ M., PONTES J. K., JACK D., BAKTASHMOTLAGH M., ERIKSSON A. P.: Deep level sets: Implicit surface representations for 3D shape inference. *arXiv:1901.06802* (2019). 3

[MQCJ18] MURALI V., QI L., CHAUDHURI S., JERMAINE C.: Neural Sketch Learning for Conditional Program Generation. In *International Conference on Learning Representations (ICLR)* (2018). 10

[MSK10] MERRELL P., SCHKUFZA E., KOLTUN V.: Computer-generated residential building layouts. In *Annual Conference on Computer Graphics and Interactive Techniques Asia (SIGGRAPH Asia)* (2010). 14, 15

[MSL*11] MERRELL P., SCHKUFZA E., LI Z., AGRAWALA M., KOLTUN V.: Interactive Furniture Layout Using Interior Design Guidelines. In *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (2011). 2, 15

[MSSH14] MAJEROWICZ L., SHAMIR A., SHEFFER A., HOOS H. H.: Filling your shelves: Synthesizing diverse style-preserving artifact arrangements. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 20*, 11 (2014), 1507–1518. 15

[MTG*13] MENON A., TAMUZ O., GULWANI S., LAMPSON B., KALAI A.: A Machine Learning Framework for Programming by Example. In *International Conference on Machine Learning (ICML)* (2013). 10

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. In *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (2006). 2, 12

[MZC*19] MO K., ZHU S., CHANG A. X., YI L., TRIPATHI S., GUIBAS L. J., SU H.: PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 19

[MZWVG07] MÜLLER P., ZENG G., WONKA P., VAN GOOL L.: Image-based procedural modeling of facades. In *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (2007). 2

[NAW*19] NANDI C., ANDERSON A., WILLSEY M., WILCOX J. R., DARULOVA E., GROSSMAN D., TATLOCK Z.: Using e-graphs for CAD parameter inference. *arXiv:1909.12252* (2019). 17

[NLX18] NIU C., LI J., XU K.: Im2Struct: Recovering 3D shape structure from a single RGB image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). 14

[PFS*19] PARK J. J., FLORENCE P., STRAUB J., NEWCOMBE R., LOVEGROVE S.: DeepSDF: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 3

[PL96] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants*. Springer-Verlag, Berlin, Heidelberg, 1996. 2, 12

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. In *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (2001). 2

[PMS*17] PARISOTTO E., MOHAMED A.-R., SINGH R., LI L., ZHOU D., KOHLI P.: Neuro-Symbolic Program Synthesis. In *International Conference on Learning Representations (ICLR)* (2017). 10

[PW14] PAIGE B., WOOD F.: A Compilation Target for Probabilistic Programming Languages. In *International Conference on Machine Learning (ICML)* (2014). 7

[QSMG17] QI C. R., SU H., MO K., GUIBAS L. J.: PointNet: deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). 3

[QYSG17] QI C. R., YI L., SU H., GUIBAS L. J.: PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NeurIPS)* (2017). 3

[RBSB18] RANJAN A., BOLKART T., SANYAL S., BLACK M. J.: Generating 3D Faces Using Convolutional Mesh Autoencoders. In *European Conference on Computer Vision (ECCV)* (2018). 3

[RDF16] REED S., DE FREITAS N.: Neural Programmer-Interpreters. In *International Conference on Learning Representations (ICLR)* (2016). 10

[RJT18] RITCHIE D., JOBALIA S., THOMAS A.: Example-based authoring of procedural modeling programs with structural and continuous variability. In *Annual Conference of the European Association for Computer Graphics (EuroGraphics)* (2018). 7, 12

[RLGH15] RITCHIE D., LIN S., GOODMAN N. D., HANRAHAN P.: Generating Design Suggestions under Tight Constraints with Gradient-based Probabilistic Programming. In *Annual Conference of the European Association for Computer Graphics (EuroGraphics)* (2015). 7

[RMGH15] RITCHIE D., MILDENHALL B., GOODMAN N. D., HANRAHAN P.: Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 1–11. 2

[ROUG17] RIEGLER G., OSMAN ULUSOY A., GEIGER A.: OctNet: Learning deep 3D representations at high resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). 3

[RWaL19] RITCHIE D., WANG K., AN LIN Y.: Fast and Flexible Indoor Scene Synthesis via Deep Convolutional Generative Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 8, 16

[SCH*16] SAVVA M., CHANG A. X., HANRAHAN P., FISHER M., NIEÃ§NER M.: PiGraphs: Learning interaction snapshots from observations. *ACM Transactions on Graphics (TOG) 35*, 4 (2016). 15

[SGF16] SHARMA A., GRAU O., FRITZ M.: VConv-DAE: Deep Volumetric Shape Learning without Object Labels. In *European Conference on Computer Vision Workshops (ECCV Workshop)* (2016). 3

[SGL*18] SHARMA G., GOYAL R., LIU D., KALOGERAKIS E., MAJI S.: CSGNet: Neural Shape Parser for Constructive Solid Geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). 17

[SHW*17] SOLTANI A. A., HUANG H., WU J., KULKARNI T. D., TENENBAUM J. B.: Synthesizing 3D Shapes via Modeling Multi-View Depth Maps and Silhouettes with Deep Generative Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). 3

[SKAG15] SUNG M., KIM V. G., ANGST R., GUIBAS L.: Data-driven structural priors for shape completion. *ACM Transactions on Graphics (TOG) 34*, 6 (2015), 175:1–175:11. 14

[SLNM11] SOCHER R., LIN C. C., NG A. Y., MANNING C. D.: Parsing natural scenes and natural language with recursive neural networks. In *International Conference on Machine Learning (ICML)* (2011). 9, 16

[SMKL15] SU H., MAJI S., KALOGERAKIS E., LEARNED-MILLER E. G.: Multi-view convolutional neural networks for 3D shape recognition. In *IEEE International Conference on Computer Vision (ICCV)* (2015). 3

[SPW*13] SOCHER R., PERELYGIN A., WU J., CHUANG J., MANNING C., NG A., POTTS C.: Recursive deep models for semantic compositionality over a sentiment treebank. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2013). 9

[SSK*17] SUNG M., SU H., KIM V. G., CHAUDHURI S., GUIBAS L.: ComplementMe: Weakly-supervised component suggestions for 3D modeling. In *Annual Conference on Computer Graphics and Interactive Techniques Asia (SIGGRAPH Asia)* (2017). 4, 12

[TDB17] TATARCHENKO M., DOSOVITSKIY A., BROX T.: Octree Generating Networks: Efficient Convolutional Architectures for High-Resolution 3D Outputs. In *IEEE International Conference on Computer Vision (ICCV)* (2017). 3

[TGLX18] TAN Q., GAO L., LAI Y.-K., XIA S.: Variational Autoencoders for Deforming 3D Mesh Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). 3

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis Procedural Modeling. *ACM Transactions on Graphics (TOG) 30*, 2 (2011). 2

[TLS*19] TIAN Y., LUO A., SUN X., ELLIS K., FREEMAN W. T., TENENBAUM J. B., WU J.: Learning to Infer and Execute 3D Shape Programs. In *International Conference on Learning Representations (ICLR)* (2019). 5, 10, 17, 18, 19

[TSG*17] TULSIANI S., SU H., GUIBAS L. J., EFROS A. A., MALIK J.: Learning Shape Abstractions by Assembling Volumetric Primitives. In

IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017). 17, 18

[TYK*12] TALTON J. O., YANG L., KUMAR R., LIM M., GOODMAN N. D., MECH R.: Learning design patterns with Bayesian grammar induction. In *ACM Symposium on User Interface Software and Technology (UIST)* (2012). 12

[VDOKK16] VAN DEN OORD A., KALCHBRENNER N., KAVUKCUOGLU K.: Pixel recurrent neural networks. In *International Conference on Machine Learning (ICML)* (2016). 8

[WFT*19] WU W., FU X.-M., TANG R., WANG Y., QI Y.-H., LIU L.: Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (TOG) 38*, 6 (2019). 14, 16

[WLG*17] WANG P.-S., LIU Y., GUO Y.-X., SUN C.-Y., TONG X.: O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 72. 3

[WLW*19] WANG K., LIN Y.-A., WEISSMANN B., SAVVA M., CHANG A. X., RITCHIE D.: PlanIt: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG) 38*, 4 (2019), 132. 1, 4, 10, 16, 17

[WSCR18] WANG K., SAVVA M., CHANG A. X., RITCHIE D.: Deep Convolutional Priors for Indoor Scene Synthesis. In *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (2018). 4, 16

[WSG11] WINGATE D., STUHLMÜLLER A., GOODMAN N. D.: Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2011). 7

[WSH*18] WANG H., SCHOR N., HU R., HUANG H., COHEN-OR D., HUANG H.: Global-to-local generative model for 3D shapes. *ACM Transactions on Graphics (TOG) 37*, 6 (2018), 214:1–214:10. 4, 13

[WSK*15] WU Z., SONG S., KHOSLA A., YU F., ZHANG L., TANG X., XIAO J.: 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015). 2, 3

[WTB*18] WANG C., TATWAWADI K., BROCKSCHMIDT M., HUANG P.-S., MAO Y., POLOZOV O., SINGH R.: Robust Text-to-SQL Generation with Execution-Guided Decoding. *arXiv:1807.03100* (2018). 10

[WWL*19] WU Z., WANG X., LIN D., LISCHINSKI D., COHEN-OR D., HUANG H.: SAGNet: Structure-aware generative network for 3D-shape modeling. *ACM Transactions on Graphics (TOG) 38*, 4 (2019), 91:1–91:14. 19

[WXL*11] WANG Y., XU K., LI J., ZHANG H., SHAMIR A., LIU L., CHENG Z., XIONG Y.: Symmetry hierarchy of man-made objects. *Computer Graphics Forum (CGF)* (2011). 4, 5, 13, 14

[WZL*18] WANG N., ZHANG Y., LI Z., FU Y., LIU W., JIANG Y.-G.: Pixel2Mesh: Generating 3D mesh models from single RGB images. In *European Conference on Computer Vision (ECCV)* (2018). 3

[WZX*16] WU J., ZHANG C., XUE T., FREEMAN W. T., TENENBAUM J. B.: Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In *Advances in Neural Information Processing Systems (NeurIPS)* (2016). 3

[XCF*13] XU K., CHEN K., FU H., SUN W.-L., HU S.-M.: Sketch2Scene: Sketch-based co-retrieval and co-placement of 3D models. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 123. 14

[XMZ*14] XU K., MA R., ZHANG H., ZHU C., SHAMIR A., COHEN-OR D., HUANG H.: Organizing heterogeneous scene collection through contextual focal points. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), Article 35. 4

[YCHK15] YUMER M. E., CHAUDHURI S., HODGINS J. K., KARA L. B.: Semantic shape editing using deformation handles. *ACM Transactions on Graphics (TOG) 34* (2015). 14

[YFST18] YANG Y., FENG C., SHEN Y., TIAN D.: FoldingNet: Point Cloud Auto-encoder via Deep Grid Deformation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). 3

[YKC*16] YI L., KIM V. G., CEYLAN D., SHEN I.-C., YAN M., SU H., LU C., HUANG Q., SHEFFER A., GUIBAS L.: A scalable active framework for region annotation in 3D shape collections. In *Annual Conference on Computer Graphics and Interactive Techniques Asia (SIGGRAPH Asia)* (2016). 11

[YSS*17] YI L., SHAO L., SAVVA M., HUANG H., ZHOU Y., WANG Q., GRAHAM B., ENGELCKE M., KLOKOV R., LEMPITSKY V. S., GAN Y., WANG P., LIU K., YU F., SHUI P., HU B., ZHANG Y., LI Y., BU R., SUN M., WU W., JEONG M., CHOI J., KIM C., GEETCHANDRA A., MURTHY N., RAMU B., MANDA B., RAMANATHAN M., KUMAR G., PREETHAM P., SRIVASTAVA S., BHUGRA S., LALL B., HÄNE C., TULSIANI S., MALIK J., LAFER J., JONES R., LI S., LU J., JIN S., YU J., HUANG Q., KALOGERAKIS E., SAVARESE S., HANRAHAN P., FUNKHOUSER T. A., SU H., GUIBAS L. J.: Large-scale 3D shape reconstruction and segmentation from ShapeNet Core55. *arXiv:1710.06104* (2017). 11

[YYR*18] YOU J., YING R., REN X., HAMILTON W. L., LESKOVEC J.: GraphRNN: A deep generative model for graphs. In *International Conference on Machine Learning (ICML)* (2018). 4

[YYT*11] YU L.-F., YEUNG S. K., TANG C.-K., TERZOPOULOS D., CHAN T. F., OSHER S.: Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG) 30*, 4 (2011), 86:1–12. 2, 15

[YYT16] YU L.-F., YEUNG S. K., TERZOPOULOS D.: The Clutterpalette: An interactive tool for detailing indoor scenes. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 22*, 2 (2016), 1138–1148. 15

[YYW*12] YEH Y.-T., YANG L., WATSON M., GOODMAN N. D., HANRAHAN P.: Synthesizing Open Worlds with Constraints Using Locally Annealed Reversible Jump MCMC. In *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (2012). 2

[YYY*16] YAN X., YANG J., YUMER E., GUO Y., LEE H.: Perspective transformer nets: Learning single-view 3D object reconstruction without 3D supervision. In *Advances in Neural Information Processing Systems (NeurIPS)* (2016). 19

[ZHG*16] ZHAO X., HU R., GUERRERO P., MITRA N. J., KOMURA T.: Relationship templates for creating scene variations. *ACM Transactions on Graphics (TOG) 35*, 6 (2016), 1–13. 15

[ZW18] ZOHAR A., WOLF L.: Automatic Program Synthesis of Long Programs with a Learned Garbage Collector. In *Advances in Neural Information Processing Systems (NeurIPS)* (2018). 10

[ZWK19] ZHOU Y., WHILE Z., KALOGERAKIS E.: SceneGraphNet: Neural message passing for 3D indoor scene augmentation. In *IEEE International Conference on Computer Vision (ICCV)* (2019). 16

[ZXC*18] ZHU C., XU K., CHAUDHURI S., YI R., ZHANG H.: SCORES: Shape composition with recursive substructure priors. *ACM Transactions on Graphics (TOG) 37*, 6 (2018), 211:1–211:14. 13, 14

[ZYM*18] ZHANG Z., YANG Z., MA C., LUO L., HUTH A., VOUGA E., HUANG Q.: Deep generative modeling for scene synthesis via hybrid representations. *arXiv:1808.02084* (2018). 16

[ZYY*17] ZOU C., YUMER E., YANG J., CEYLAN D., HOIEM D.: 3D-PRNN: Generating Shape Primitives with Recurrent Neural Networks. In *IEEE International Conference on Computer Vision (ICCV)* (2017). 17

[ZZZ*18] ZHU J.-Y., ZHANG Z., ZHANG C., WU J., TORRALBA A., TENENBAUM J., FREEMAN B.: Visual object networks: Image generation with disentangled 3D representations. In *Advances in Neural Information Processing Systems (NeurIPS)* (2018). 19