Order-Aware Generative Modeling Using the *3D-Craft* Dataset

Zhuoyuan Chen*, Demi Guo*, Tong Xiao*, Saining Xie, Xinlei Chen,
Haonan Yu, Jonathan Gray, Kavya Srinet, Haoqi Fan, Jerry Ma, Charles R. Qi, Shubham Tulsiani,
Arthur Szlam, and C. Lawrence Zitnick
Facebook AI Research, Menlo Park, CA

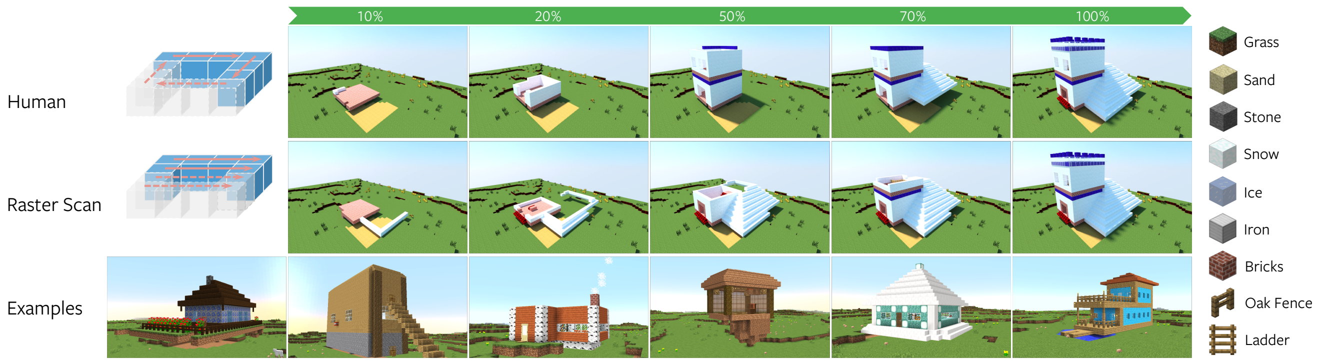

Figure 1. We present *3D-Craft*, a new dataset of diverse houses built from scratch by human players in the game of Minecraft. The first and second row illustrate the difference between a recorded human action sequence and a predefined raster scan order for building a house. The human order information enables us to learn order-aware generative models to predict actions in a more intuitive, human-like manner.

Abstract

Research on 2D and 3D generative models typically focuses on the final artifact being created, e.g., an image or a 3D structure. Unlike 2D image generation, the generation of 3D objects in the real world is commonly constrained by the process and order in which the object is constructed. For instance, gravity needs to be taken into account when building a block tower.

In this paper, we explore the prediction of ordered actions to construct 3D objects. Instead of predicting actions based on physical constraints, we propose learning through observing human actions. To enable large-scale data collection, we use the Minecraft[1] environment. We introduce *3D-Craft*, a new dataset of 2,500 Minecraft houses each built by human players sequentially from scratch. To learn from these human action sequences, we propose an order-aware 3D generative model called VoxelCNN. In contrast to other 3D generative models which either have no explicit order (*e.g.* holistic generation with 3D-GAN [35]), or follow a simple heuristic order (*e.g.* raster-scan), VoxelCNN is trained to imitate human building order with spatial awareness. We also transferred the order to other dataset such as ShapeNet[10]. The *3D-Craft* dataset, models, and benchmark system will be made publicly available, which may inspire new directions for future research exploration. https://github.com/facebookresearch/VoxelCNN.

*equal contribution
[1]Minecraft features: ©Mojang Synergies AB included courtesy of Mojang AB

1. Introduction

Generative modeling is a fundamental problem in machine learning and has a long history in computer vision. Numerous approaches have been proposed for 2D image generation, including autoregressive models [32, 31, 25] and Generative Adversarial Networks (GANs) [40, 6]. Along with the success of 2D models, there is an increasing interest in generative modeling of 3D objects in the vision community, with many applications such as multi-view 3D reconstruction [29], 3D editing [23], and probabilistic generative modeling [35].

Unlike 2D image generative models, a potential goal of 3D generative models is to create real world physical objects. The creation of a physical objects not only requires the final design of the object, but also *the order and process used to create it.* For instance, building an IKEA furniture requires one to know the assembly order, building a house must conform to gravity to build sequentially the foundation, pillars, walls, roof, etc. 3D printing technology also imposes orderings and restrictions on the construction of objects.

A question then naturally follows: how can we learn the orderings necessary to construct a 3D object? One way is to directly simulate the physics of the environment, which is made difficult by the endless range of possible constraints. An alternative approach is to learn from observations: we can aim to imitate human behavior, since human order implicitly conforms to the physical constraints of the world. Unfortunately, gathering large amounts of human observations can be incredibly difficult.

This paper, following the second approach, explores order-aware 3D generative models on the Minecraft platform. While Minecraft is a simplified synthetic environment, we hypothesize that studying problems in this domain may shed light on effective approaches for generating real world physical objects. Similar to real world objects, Minecraft houses have significant ambiguity in their construction, have many material types, and contain numerous sub-parts, e.g., roofs, walls, doors, windows, etc. We collect a large dataset of houses each built by humans sequentially from scratch. The houses are constructed of coarse 3D blocks, or voxels, of different materials (wood, stone, glass, etc.). Our dataset, dubbed *3D-Craft*, contains more than 2,500 houses with objects built by 2.8 million building steps of 256 materials from more than 200 unique human players. To our knowledge, this is the first 3D volumetric dataset with order information. As a popular game platform with more than 91 million monthly players, Minecraft offers a unique opportunity to collect a large dataset of this type.

With *3D-Craft*, we not only focus on the final built houses, but more importantly, understand how to generate a 3D object in a natural order, and how to recover the natural order given the final 3D object. Inspired by successful autoregressive approaches for generating 2D images, such as PixelRNN [31] and PixelCNN [32] , we propose VoxelCNN to build a 3D object voxel-by-voxel for our sequential 3D generation problem. Conditioned on the previous building sequence, our model recovers the partial 3D object, employing a Convolutional Neural Network (CNN) to encode spatial structure, and predicts a distribution over the next voxel to be placed, including both the location and the material of the voxel. In contrast to [31, 32] that generate pixels in a predefined raster-scan order, our VoxelCNN is trained to imitate natural human building order, which is crucial for the sequential 3D generation problem through our experiments.

We propose several metrics to evaluate VoxelCNN. Unlike the qualitative evaluation used in many other generative tasks where only the final product is of interest, with order information, our metrics quantitatively measure how well the model predictions match human actions, how many voxels can be placed before a mistake is made, and how many times a human needs to correct the model if it were to build the entire house. These metrics help us better understand the sequential generation process.

2. Related Work

3D Datasets. Numerous 3D datasets have been built or collected for research on 3D objects. These include the use of CAD models [10, 37], 3D objects aligned to images with anchor points [39], template alignment [38], and 3D printing models [41]. Our work is also related to datasets that attempt to model 3D scenes, such as the SUNCG [27] and Matterport3D [9] datasets. These have been used for a variety of embodied QA [11, 8, 19] and navigation tasks [26, 36, 8].

In this paper, we explore the task of building 3D environments. The *3D-Craft* dataset is unique in that it contains the order in which humans created the the 3D houses, and each block has an associated type (rock, wood, glass, etc.). However, *3D-Craft* is less visually realistic than both the SUNCG [27] and Matterport3D [9] datasets.

3D Modeling. There has been impressive progress in 3D synthesis and reconstruction over the decades, mostly based on either parametric morphable models [5, 2] or part-based template learning [12, 20]. Recent advance in deep learning has shown promising improvement in various 3D-vision models and applications, including synthesis [35], reconstruction [34, 42], part-based analysis [28] and interactive editing [23].

Autoregressive Models. There have been many 2D autoregressive methods such as [22, 30, 31, 32, 25, 17]. Though flexible and expressive, these approaches tend to be slow due to its sequential execution dependency (requiring width $\times$ height steps), and the issue becomes more severe in 3D domain (width $\times$ length $\times$ height). Our approach makes use of the sparsity of the 3D occupancy and only predicts content on occupied voxels.

Order-aware Datasets. There have been datasets with generative order annotation including hand-written characters [21] and buildings [24]; [21] records stroke-by-stroke human order, while [24] contains top-down grammar for building models.

Order-aware Generative Models. [33] shows that the order of data organization matters significantly in sequence-to-sequence (seq2seq) modeling. There has been a series of recent works proposed to generate strokes on a white canvas, such as Bayesian Program Learning [21], recurrent VAE [14], 3D-PRNN [42] and reinforcement learning [13]. In [14, 13], there is no strict constraint to follow human orders, and quality of generation is evaluated at the last step when generation ends. Since action-level supervision is provided, these approaches tend to be more general but also more complex and harder to train. [24] designed a shape grammar for procedural modeling of CG architectures, which builds the rough structures followed by details such as windows and doors.

Game Platforms for AI. In recent years, a range of game platforms for AI agents have been proposed. These focus on a variety of tasks, such as reasoning and embodied Question Answering [11, 19, 36, 16], reinforcement learning with defined goals [4, 7, 1, 18, 16], and visual control and navigation [16, 18, 19, 26, 8, 16, 3]. We build our task in the Minecraft setting. Similar to the Malmo project [16], which is also built on Minecraft, we view Minecraft as an attractive platform for studying open-ended creative tasks.

3. *3D-Craft* Dataset

In this section, we introduce the Minecraft game and the *3D-Craft* dataset.

3.1. Minecraft

Minecraft is a popular open-world sandbox video game developed by Mojang [2]. The game allows players to explore and manipulate a procedurally generated world. Players can place and destroy blocks of different material types in a 3D grid. Minecraft, particularly in its creative mode setting, has no win condition and encourages players to be creative.

Minecraft is a closed-source game but several open source community-built projects exist, including clones of the game (*e.g.* Cuberite[3] and Craft[4]). To enable using Minecraft for artificial intelligence research, Project Malmo [16] has provided a platform built on top of Minecraft that allows researchers to control a Minecraft bot. For our paper, we leverage the Cuberite server to collect the data. Cuberite is an open-sourced Minecraft-compatible game server with extensive plugins for players and developers.

3.2. Data Collection

We used crowd sourcing to collect examples of humans building houses in Minecraft. Each user is asked to build a house on a fixed time budget (30 minutes), without any additional guidance or instruction. Every action of the user is recorded using the Cuberite server.

The data collection was performed in Minecraft's creative mode, where the user is given unlimited resources, has access to all material block types and can freely move in the game world (*e.g.* flying through the air). The action space of the environment is thus straight-forward: moving in x-y-z dimensions, choosing a block type, and placing or breaking a block. Any placed blocks must be attached to a neighboring block, i.e., blocks cannot be placed in the air.

Notably, there are hundreds of different block types someone could use to build a house, including different kinds of wood, stone, dirt, sand, glass, metal, ice, to list a few. We show some materials in Figure 1. An empty voxel is considered as a special block type "air" (block id=0).

We record sequences of atomic building actions for each user, at each step using the following format:

```
[t, userid, [x, y, z],
    [block-id, meta-id], "P"/"B"]
```

where the time-stamp t is always in monotonically increasing order; $[x_t, y_t, z_t]$ is the absolute coordinate with respect to the world origin in Minecraft; "P" and "B" refers to placing a new block and breaking (destroying) an existing block; each house is built by a single player in our data collection process with a unique user-id.

3.3. Data Cleaning

To encourage diversity and creativity in our data collection pipeline, we intentionally impose no restrictions on the house crafting task except for the time allowed for building. However, the raw data collected from human players needs to be pre-processed based on a few observations. Firstly, a player might change their mind while designing the house and "undo" a build action by removing an existing block, e.g., remove some blocks on a wall to make room for a window. Secondly, a few constructions are caves or underground shelters constructed by destroying blocks in the ground or a mountainside. Finally, players might build arbitrarily large houses in the open world of Minecraft or create disjoint structures over large areas.

We clean up the raw data in *3D-Craft* by the following preprocessing steps: 1) If multiple actions are taken on the same location, we only keep the action with the largest time-stamp. 2) We remove cave homes, underground shelters or other excavated houses from our dataset. 3) We perform connected component analysis on houses and only the largest connected structure is kept.

All statistics, experimental setup, and evaluation results in the following sections are reported using the cleaned data.

3.4. Dataset Statistics

In this section, we describe the statistics of the *3D-Craft* dataset. Specifically, we analyze several properties of the fully-built houses and the player action sequences that created them. The houses were created by approximately 200 unique human players. The number of houses built per player is shown in Figure 2 (f).

We begin by examining how many blocks it takes to craft a completed house from scratch. We show the histogram

[2]https://mojang.com/category/minecraft/
[3]https://github.com/cuberite
[4]https://github.com/fogleman/Craft

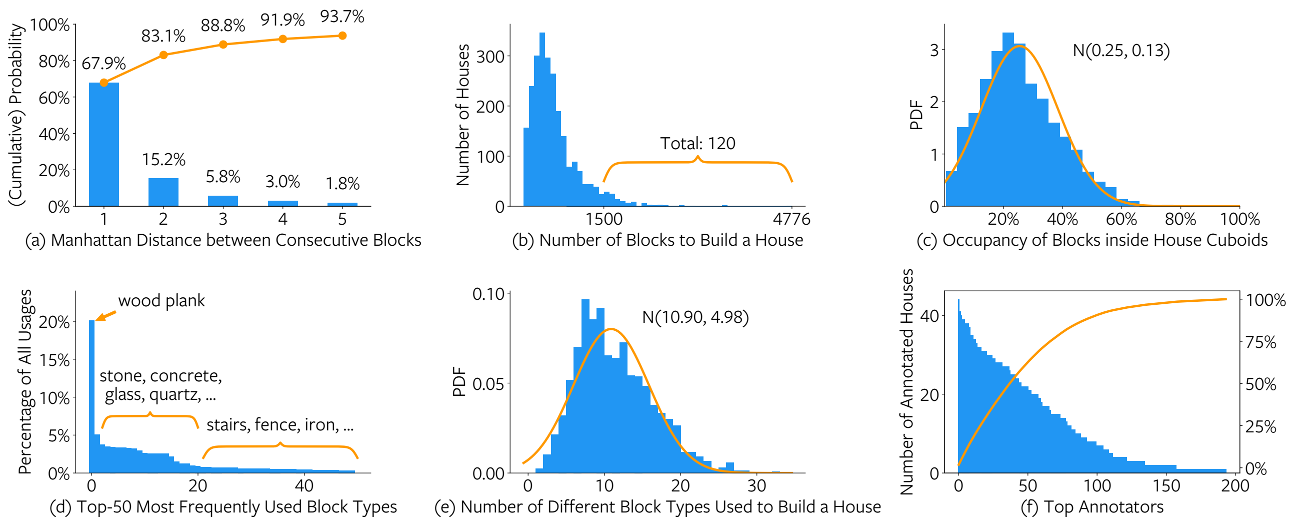

Figure 2. Dataset statistics. (a) 67.9% of all the blocks are placed within a 1 block distance of the previously placed one. (b) On average, each house has 635 blocks, but there are 120 houses built with more than 1,500 blocks. (c) House blocks are sparse in the 3D space. On average, only 25% of the cuboid voxels in a house are occupied by human-built blocks. (d) Most frequently used block types. Wood plank is used by 20% of all the 1.5M human-built blocks. (e) On an average, each house is built by 10.9 different block types. (f) Around 200 annotators contributed to the 2.5K houses. The most productive annotator built more than 40 houses.

of house sizes represented by the number of blocks in Figure 2 (b). We can observe that the distribution is single-mode and heavy-tailed, with mean 635 and median 526. We also observe that players tend to work with multiple types of blocks to build the houses, with an average of 10.9 different materials being used per house, shown in Figure 2 (e). The block types used also have a heavy-tailed distribution as shown in Figure 2 (d). Block types such as wood plank and stone are commonly used, while fences, stairs, iron, *etc.* are used less.

Finally, we show the properties of sequential block placement in Figure 2 (a). Under L1 (Manhattan) distance, approximately 70% of blocks are placed within 1 block of the previously placed block. 93.7% of blocks are placed within 5 blocks. This is consistent with our intuition: people tend to finish up a complete and structured sub-part, such as an entire wall, before moving onto another. Large jumps generally occur only when players jump between one sub-part to another, *e.g.*, moving from a roof to a window. The reader can refer to videos in our supplemental materials for recorded house building action sequences. This spatial locality property makes *3D-Craft* a suitable test bed for ordered generation tasks, as discussed in the next section.

4. Order-aware 3D Generative Modeling

In this section, we formalize the problem of order-aware generative modeling for *3D-Craft* objects, and introduce our VoxelCNN model to solve the problem.

4.1. Problem Definition

A house A is generated by a sequence of T actions $A = \{a_1, a_2, ..., a_T\}$, where each action $a_t = \{\lambda_t, b_t\}$ places a new block at position $\lambda_t = \{x_t, y_t, z_t\}$ using block type b_t. We use $a_{t:t+k}$ to denote the action sub-sequence $\{a_t, a_{t+1}, \ldots, a_{t+k}\}$. Our goal is to predict the next action a_{t+1} given $a_{1:t}$.

4.2. VoxelCNN with Natural Human Order

VoxelCNN models the joint distribution of actions over A as the product of conditional distributions, where action a_i is a single block (position and block type):

$$p(A) = \prod_{t=0}^{T-1} p(a_{t+1}|a_{1:t}) \tag{1}$$

Every block therefore depends on all the blocks placed before it, in *natural human order*. For each action a_{t+1}, we let the block type b_{t+1} depend on the position λ_{t+1} as:

$$\begin{aligned} p(a_{t+1}|a_{1:t}) &= p(\lambda_{t+1}, b_{t+1}|a_{1:t}) \\ &= p(\lambda_{t+1}|a_{1:t})p(b_{t+1}|\lambda_{t+1}, a_{1:t}) \end{aligned} \tag{2}$$

The intuition is that the *what* depends on the *where*, which is inspired by Conditional PixelCNN [32] on 2D images, where three channels R, G, B are modeled successively. However, the PixelCNN follows heuristic raster-scan order, while VoxelCNN targets on learning natural human order.

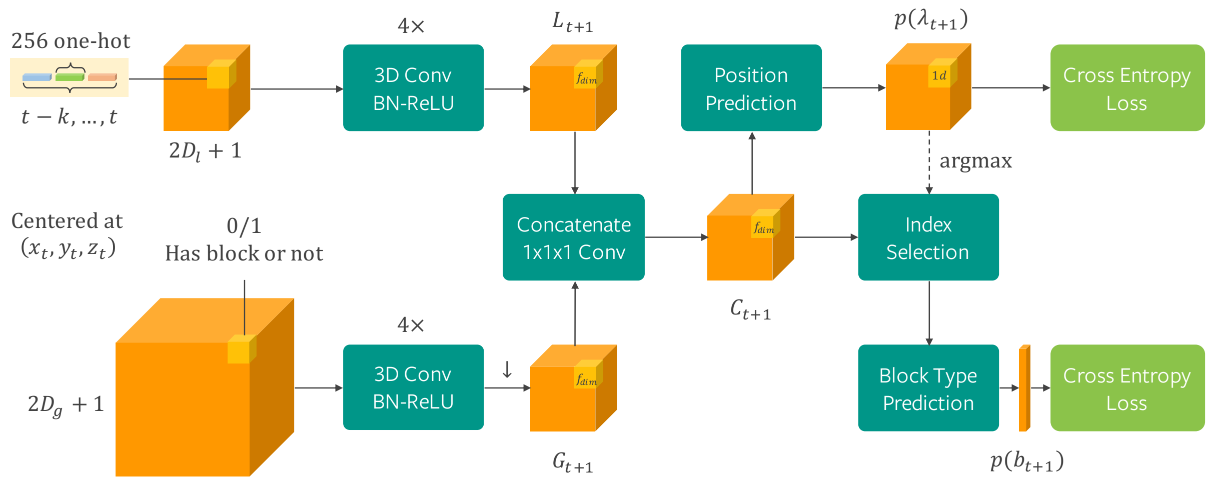

Figure 3. VoxelCNN architecture. Centered at the last action, the input for the local branch is a concatenation of $256 \times (2D_l+1)^3$ one-hot vectors for the past $k+1$ history actions, while for the global branch it is a $1 \times (2D_g+1)^3$ binary tensor. We pass both inputs through 4 layers of 3D Convolution-BatchNorm-ReLU modules, and then concatenate and transform them to the feature tensor C_{t+1}. The $(2D_l+1)^3$ probability of the position $p(\lambda_{t+1}|a_{1:t})$ is first predicted , and then 256-d probability of materials $p(b_{t+1}|\lambda_{t+1}, a_{1:t})$ is predicted for the most probable position.

We model $p(a_{t+1}|a_{1:t})$ by a CNN f_θ with parameters θ. As shown in Figure 3, the network first recovers the state of the 3D object s_t in voxels based on the action sequence $a_{1:t}$. Then it centers on the last placed block, encoding the multi-resolution spatial contexts, and predicts both $p(\lambda_{t+1}|a_{1:t})$ and $p(b_{t+1}|\lambda_{t+1}, a_{1:t})$. Note that $p(\lambda_{t+1}|a_{1:t})$ is a distribution over the neighborhood relative to the last voxel λ_t.

To capture both the global design and detailed local structure, a two-stream framework is proposed to encode multi-resolution spatial contexts. The input state s_t consists of two 3D patches – the local context $s_{t,l}$ and the global context $s_{t,g}$, with radius D_l and D_g respectively. As shown in Figure 3, features are extracted from $s_{t,l}$ and $s_{t,g}$ separately with a late feature fusion.

Local Encoding. A 3D local neighborhood of $s_{t,l}$ is represented by a $256 \times (2D_l+1)^3$ tensor, which consists of one-hot vector of block types of all voxels within the D_l-neighborhood of the last block built at time t. We then apply multiple 3D-convolution layers to obtain a final local representation L_{t+1} of size $f_{dim} \times (2D_l+1)^3$.

Global Encoding. To capture the overall design of the house, we encode the global state $s_{t,g}$ with a much larger radius D_g than the local state $s_{t,l}$ (in our experiments, we set $D_g = 10$ and $D_l = 3$). Compared with local context $s_{t,l}$, the global context $s_{t,g}$ only contains binary occupancy (air/non-air), which focuses on the overall geometry of the house and helps avoid overfitting during training. An additional max-pooling layer is applied to reduce the size of global representation G_{t+1} to $f_{dim} \times (2D_l+1)^3$ as well.

Late Feature Fusion. We then concatenate the local representation L_{t+1} and global representation G_{t+1} along the feature channels, and apply a $1 \times 1 \times 1$ 3D-convolution layer to obtain the final contextual representation C_{t+1} of $f_{dim} \times (2D_l+1)^3$.

Temporal Information. It is also of interest to explicitly model longer-term temporal information in encoding, since consecutive actions tend to be spatially correlated. We propose to concatenate the local house states $s_{t,l}, s_{t-1,l}, \dots, s_{t-k,l}$ together into $\hat{s}_{t,l}$, and then feed $\hat{s}_{t,l}$ as the input to the *local encoding* module.

Factorized Prediction. Based on the final representation C_{t+1}, we apply a $1 \times 1 \times 1$ 3D-convolution layer on top to predict the position $p(\lambda_{t+1}|a_{1:t})$ as a tensor of $(2D_l+1)^3$, followed by a softmax layer. For block type b_{t+1}, we take the f_{dim} vector in C_{t+1} at either ground truth location (training), or greedy predicted location by argmax (test), and use a linear layer to obtain a 256-d vector, followed by a softmax layer. We apply cross-entropy loss to train both predictions.

Note that the current prediction could be limited to within a local neighborhood of the last block. However, setting $D_l = 3$ already covers over 90% of all the ground truth cases. It can be extended by setting larger D_l or using pyramid-like hierarchical predictions.

5. Evaluation Metrics

Evaluating generative models quantitatively is known to be non-trivial. However, the ground truth sequential ordering $a_{1:T}$ in *3D-Craft* allows us to evaluate sequential prediction $\hat{a}_{1:T}$ with four quantitative metrics, measuring the quality of both the final house and the generation process:

1) accuracy of the next block placement; 2) action sequence perplexity; 3) number of consecutive correct actions before making a mistake; and 4) number of mistakes that need to be corrected in order to complete a house.

Accuracy of next step (ACC@N): We measure how well a method can predict the next block (position and block type) that matches natural human order. Since it could be ambiguous for human actions in some cases as well, we relax the metrics to evaluate whether the predicted block is in the next $N \in \{1, 5, 10\}$ moves made by a human. This allows for some flexibility in the local ordering of block placement. Formally, we test if the prediction $\hat{a}_{t+1}$ matches any one of the ground truth actions $a_{t+1:t+N}$.

Perplexity: We can also measure the performance of a generative model by Perplexity (ppl):

$$\log_2 \text{ppl} = \mathbb{E}[-log(p(\hat{a}_{t+1} = a_{t+1}|a_{1:t}))]$$

Perplexity measures the quality of the immediate behavior-cloning and only considers an action to be correct if it exactly follows the human action order.

Number of Consecutive Correct Actions (CCA): Starting from a house $X\%$ complete, we record the number of consecutive blocks the model can place $\{\hat{a}_{t+1}, \hat{a}_{t+2}, ...\}$ before it makes a mistake. An action $\hat{a}_{t+1}$ is considered correct if the placed block belongs to the finished house, *i.e.* $\hat{a}_{t+1} \in a_{t+1:T}$. For each house, we set X to be $10, 25, 50, 75, 90$, and take the average over all possible X values and houses as the final metric.

Number of Mistakes to Complete (MTC): Starting from a house X=10% complete, we record the number of mistakes made before completing the house. When the model makes a mistake, we correct the wrong prediction by an "oracle" block, which is the earliest block from the ground truth data that has not been placed yet. Since different houses might require vastly different number of blocks to complete, we further consider dividing the number of mistakes by the number of blocks for each house, and averaging across all the houses to be the Normalized MTC metric.

6. Experiments

In this section, we detail the experiment settings, comparing the proposed VoxelCNN with several baseline approaches for order-aware generation, order recovery and transferability.

6.1. Baselines

We propose several baselines to evaluate how well our model mimics the human ordering of actions, and gauge overall generation performance.

LSTM. For sequential prediction tasks, one commonly used baseline approach is Long Short-Term Memory [15]. We embed the previous actions $a_{1:t}$ into 512-d embedding space, and employ an one-layer LSTM to predict the next actions. Detailed architectures can be found in supplementary materials. In our problem, LSTM leverages the order information but does not explicitly model the spatial structure of the 3D object.

Besides, we observe that human building order may follow certain patterns, for example, continuing to build the same type of blocks in a line. We also notice that similar local structures may appear frequently, such as the windows, doors, and stairs. Therefore, we propose two intuitive non-learning baselines based on these observations:

Naive Inertia. At each step $t + 1$, place the block at $\lambda_{t+1} \leftarrow 2\lambda_t - \lambda_{t-1}$, and keep the same block type as $b_{t+1} \leftarrow b_t$.

Nearest Neighbor. We create a look-up table of 3D binary-occupancy patches for all the time steps in the training set. For each occupancy pattern, we record its most probable next-step action. At test time, for a new local patch, we find its nearest neighbor in L_2 distance and copy over the next-step action.

Inspired by PixelCNN and PixelRNN for generating 2D images, we propose another two baselines using raster-scan order, rather than natural human order, to generate 3D houses and objects.

3D PixelCNN. We extend the PixelCNN [32] to the 3D scenario by enforcing the model to predict blocks in a raster-scan order over the 3D grid. We change our VoxelCNN framework by using masked 3D convolution layers, removing position prediction head, but adding an additional block type "air" (place nothing).

Learned Raster-Scan. Given a partially built house in human order A_t, we train a VoxelCNN to predict the next unplaced block in the raster scan order instead of the human order a_{t+1}.

6.2. Implementation Details

The dataset is randomly split into 70% of the data (1750 houses; 1,074,647 steps) for training, 10% (250 houses; 159,787 steps) for validation, and 20% (500 houses; 313,265 steps) for testing.

We set hyper-parameters $D_l = 3$ and $D_g = 10$. We use four convolution layers with $f_{dim} = 16$ and kernel size of $3 \times 3 \times 3$. To encode temporal information, we stack $k = 3$ states of previous steps. Our network is trained using SGD with learning rate 0.01, Nesterov momentum 0.9, and mini-batch size of 64 for 20 epochs, and we select the model based on best ACC@1 on the validation dataset to do the final test evaluation.

6.3. Order-Aware Generation

We first evaluate different models on the task of order-aware generation (Section 4.1) with various metrics proposed in Section 5.

Comparison with Baselines. In Table 1, we compare VoxelCNN with the two groups of baseline approaches. The first group (row 1-3) shows the results of heuristic algorithms or conventional sequential model based on natural human order. While the second group (row 4-5) consists of the models based on raster-scan order. Our proposed VoxelCNN outperforms all the other baselines by a large margin, as analyzed below in more details.

In the first group, we can see that LSTM performs comparably with or even worse than the other two heuristic algorithms. As explained in Section 6.1, LSTM is designed for general sequential prediction tasks. In our problem, it leverages the building order information in a brute-force way, without explicitly modeling the spatial structure of the 3D house, which makes it hard to be trained effectively. However, VoxelCNN takes both spatial structure (locally and globally) and temporal order into consideration, which leads to superior results.

Additionally, comparing VoxelCNN with 3D PixelCNN or Learned from Raster-Scan, we can see that learning from different orders may dramatically impact the performance of generation. In our dataset, many of the chunks of blocks are semantically meaningful, such as forming a wall, pillars, doors, etc. We hypothesize that raster-scan will break the semantic continuity in generation, making the prediction less accurate.

Additional Analysis. It is of interest to study how the task difficulty varies conditioned on different percentages of building progress. As shown in Figure 4 (top), the task is much harder when the house crafting just starts (less than 10%) and almost finishes (larger than 75%), likely because the former gives too little information about the structure and the latter leaves some decoration work incomplete with much larger uncertainty and unpredictability. Also in Figure 4 (bottom), we show the distribution of $p.d.f.$ and $c.d.f.$ of CCA, which is heavy-tailed. For more ablation studies, please refer to the supplementary materials.

Qualitative Results. We display qualitative results demonstrating the sequential behavior of VoxelCNN. For a given house in the test set, we start from 50% complete and let our model predict for another 50 steps, without interrupting it even when it makes a mistake. In this scenario, undesirable wrong steps may result in error-compounding and the results may significantly deviate from the original design.

In Figure 5 we show samples of half-finished houses and compare the model results with ground truth. The first row contains five houses half-way finished; we show the progress of our model after 50 steps (middle row) and ground truth after 50 steps (third row).

We observe some interesting behaviors of our model, for example, it has the ability to add a roof on top of the walls

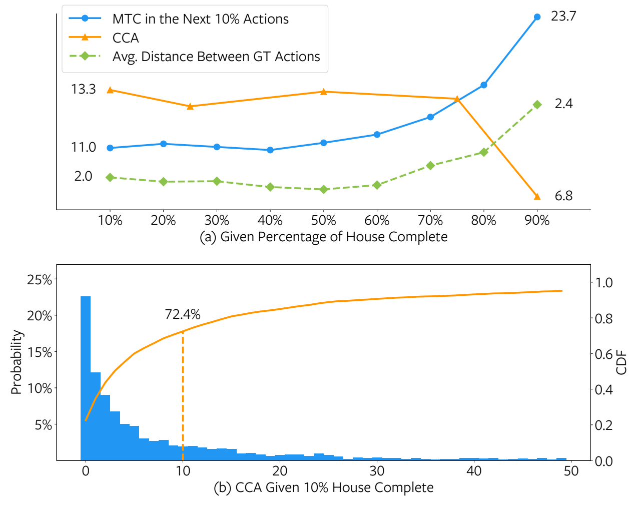

Figure 4. Evaluation results of our best VoxelCNN. (a) The model makes more mistakes when building the final stage of the house. We noticed that, towards the final stages of a house in the ground truth, both the mean and std of the distances between consecutive blocks get increased, which correlates with the MTC curve. (b) Given 10% complete houses over the dataset, the probability of CCA follows a heavy-tailed distribution.

(column 1, 2, 3). Furthermore, it sometimes mimics the way humans design by not just trivially copying the same material from the last step, but trying to switch to a different material to finish a wall (column 4). It is encouraging to see that our model performs reasonably well on half-done houses that are very distinct from one another. Admittedly, sometimes it does not perform that compelling, as shown in column 5, where it just repeatedly adds more "wall-blocks" but ignores the clear intention of finishing up with a roof and adding some doors as a human would.

6.4. Order Recovery and Transferability

VoxelCNN is shown capable of generating reasonable 3D houses voxel-by-voxel in natural order. A straightforward extension is to explore if the model can be used to recover the human building order given a final 3D house we would like to build. This order recovery problem itself is interesting and may help with tasks such as segmentation or inferring part primitives.

To this end, we make a slight modification in VoxelCNN by adding an extra input of the final object in our local encoding. We show the performance of our VoxelCNN and the Nearest Neighbor baseline that is also informed of the final object in Table 2.

It is of interest to see if the synthetic order learned from *3D-Craft* can be **transferred** to realistic 3D objects such as ShapeNet [10]. We observe that our model tends to build an

Methods	ACC@1 (%)	ACC@5 (%)	ACC@10 (%)	CCA	MTC	Normalized MTC (%)	Perplexity
LSTM	32.1	41.3	43.5	-	278.3	50.6	4.60
Naive Inertia	38.7	47.0	48.3	1.8	287.7	52.5	-
Nearest Neighbor	42.9	59.7	61.9	4.7	209.2	37.9	-
Naive VoxelCNN	-	-	-	$2.2^\dagger$	$430.5^\dagger$	-	$3.79^\dagger$
Learned Raster-Scan	31.5	49.0	52.8	2.4	295.3	53.1	4.44
VoxelCNN	**62.7**±0.17	**77.2**±0.14	**78.9**±0.15	**11.7**±0.42	**122.8**±1.07	**23.2**±.002	**3.24**±.016

Table 1. Comparisons between baseline approaches and variants of VoxelCNN. Standard deviation is measured by running our model 5 times with different random seeds. Please see Section 5 for definitions of evaluation metrics, and Section 6.1 for descriptions for each of the baseline.

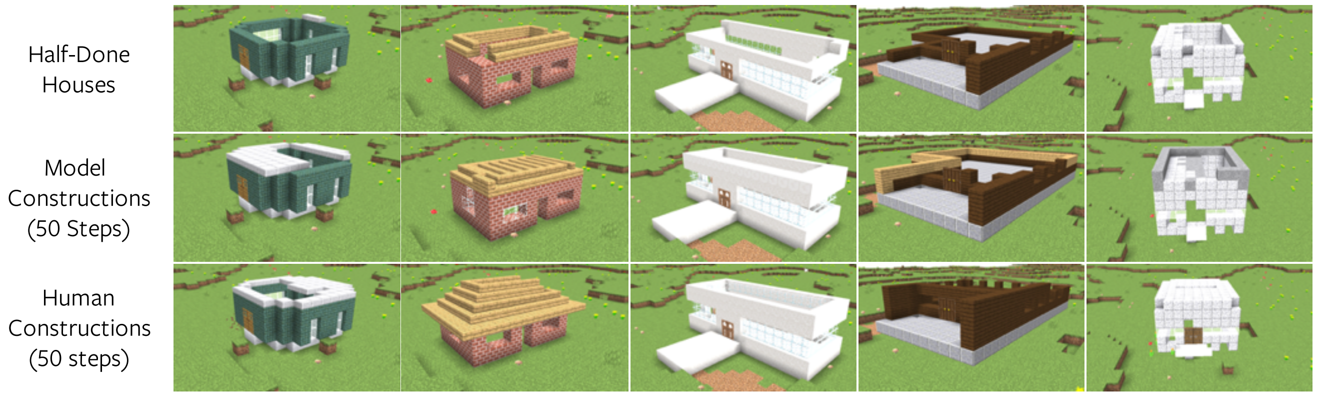
Half-Done Houses
Model Constructions (50 Steps)
Human Constructions (50 steps)

Figure 5. Sample results generated by our best model. *Top row:* houses with 50% blocks placed; *Middle row:* constructions from our model with 50 newly generated blocks; *Bottom row:* constructions from ground truth data with next 50 blocks.

Methods	ACC@1 (%)	ACC@10 (%)	Perplexity
Nearest Neighbor	43.3	62.4	-
VoxelCNN	**69.3**	**88.0**	**1.41**

Table 2. Results of order recovery from a given object. We extend the VoxelCNN to condition on an extra input of the final state of the house, and test how well it can recover the underlying sequence of human building the house.

object part by part, even though it was trained on a different domain (houses vs. chairs/tables/etc) without any parts information. Please refer to our supplemental materials for more visualizations and results.

7. Future Work and Conclusion

Several possible directions exist for future work. We currently construct houses one block at a time. An alternative is to construct houses one part, e.g., wall, roof, door, etc., at a time. We plan to add these semantic labels to the dataset in the future. Our model also assumes the next block to be placed is within a local window of the last block placed. If we used part-based building, we may be able to better predict large jumps in blocks, which, though not common, are important for whole house generation. Our dataset is based on a synthetic Minecraft world. We view this as a first step in studying the problem of constructing 3D objects in the real world. However, our simplified world does not have the same complexity as many real world problems. For instance, building with wooden blocks would result in a much larger action space.

In this paper, we study the novel problem of predicting the order in which 3D objects are constructed. We introduce a new dataset *3D-Craft* consisting of 2500 houses built by human players in Minecraft. In addition to the 3D arrangement of blocks making up the structure, we record the order in which the block were placed. We propose a VoxelCNN model that can predict the position, material and order in which to place blocks. The model may be conditioned on a predefined house, or may generate new blocks given a partially built structure. Apart from 3D object generation, the *3D-Craft* dataset may also inspire future research for indoor layout and decoration, finding 3D primitives, and human-AI collaborative creation.

References

[1] Openai universe. *https://universe.openai.com/*, 2016.

[2] Brett Allen, Brian Curless, and Zoran Popovic. The space of human body shapes: reconstruction and parameterization from range scans. *TOG*, 2003.

[3] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Vctor Valds, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab. *arxiv*, 2016.

[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *JAIR*, 2013.

[5] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. *SIGGRAPH*, 1999.

[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *ICLR*, 2019.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arxiv*, 2016.

[8] Simon Brodeur, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron Courville. Home: a household multimodal environment. *ICLR Workshop*, 2018.

[9] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *3DV*, 2017.

[10] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical report, 2015.

[11] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. *CVPR*, 2017.

[12] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. *TOG*, 2004.

[13] Iaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. *ICML*, 2018.

[14] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *ICML*, 2015.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[16] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. *IJCAI*, 2016.

[17] Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *ICML*, 2017.

[18] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *Computational Intelligence and Games*, 2016.

[19] Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: Photorealistic interactive environments for ai agents. *arxiv*, 2017.

[20] Vladislav Kreavoy, Dan Julius, and Alla Sheffer. Model composition from interchangeable components. *PG*, 2007.

[21] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 2015.

[22] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. *AISTATS*, 2011.

[23] Jerry Liu, Fisher Yu, and Thomas Funkhouser. Interactive 3d modeling with a generative adversarial network. *3DV*, 2017.

[24] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *TOG*, volume 25, pages 614–623. ACM, 2006.

[25] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications. *ICLR*, 2017.

[26] Manolis Savva, Angel X. Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. MINOS: Multimodal indoor simulator for navigation in complex environments. *arXiv:1712.03931*, 2017.

[27] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *CVPR*, 2017.

[28] Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. *CVPR*, 2017.

[29] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. *CVPR*, 2017.

[30] Benigno Uria, Iain Murray, and Hugo. Larochelle. Rnade: The real-valued neural autoregressive density estimator. *NIPS*, 2013.

[31] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *ICML*, 2016.

[32] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *NIPS*, 2016.

[33] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv:1511.06391*, 2015.

[34] Jiajun Wu, Tianfan Xue, Joseph J. Lim, Yuandong Tian, Joshua B. Tenenbaum, Antonio Torralba, and William T. Freeman. Single image 3d interpreter network. *ECCV*, 2016.

[35] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *NIPS*, 2016.

[36] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. *ICLR Workshop*, 2018.

[37] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. *CVPR*, 2015.

[38] Yu Xiang, Wonhui Kim, Wei Chen, Jingwei Ji, Christopher Choy, Hao Su, Roozbeh Mottaghi, Leonidas Guibas, and Silvio Savarese. Objectnet3d: A large scale database for 3d object recognition. *ECCV*, 2016.

[39] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. *WACV*, 2014.

[40] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. *CVPR*, 2018.

[41] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arxiv*, 2016.

[42] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks. In *Proc. ICCV*, volume 2, 2017.