Learning Layout Generation for Virtual Worlds

Weihao Cheng
ARC Lab, Tencent
whcheng@tencent.com

Ying Shan
ARC Lab, Tencent
yingsshan@tencent.com

Abstract

The emergence of the metaverse leads to rapidly rising demands for generating extensive 3D worlds. We consider that an engaging world is built upon a rational layout of multiple landuse areas (e.g., forest, meadow, farmland, etc). Towards this perspective, we propose a generative model of landuse distribution by learning geographic data. The model is based on transformer architecture that causally generates a 2D map of landuse layout, which can condition on spatial and semantic hints if one or both are provided. The model enables diverse layout generation with user controls and layout expansion by extending borders with partial inputs. To generate high-quality and satisfactory layouts, we devise a geometry objective function that supervises the model to perceive layout shapes and regularize the generations with geometric priors. We further devise a planning objective function that supervises the model to perceive progressive composition demands and suppress the generations deviating from controls. For evaluating the spatial distribution of the generations, we train an autoencoder to embed landuse layouts into vectors so that real and generated data can be compared using the Wasserstein metric inspired by Fréchet inception distance.

***Keywords:** Layout Generation, Virtual Worlds, Landuse*

1. Introduction

The explosion of interest in the metaverse [7] creates a huge demand for building 3D worlds where people and their avatars can work, shop, travel, and attend classes. For decades, computer algorithms have been developed to generate plausible 3D world components such as terrains, buildings, animals, plants, etc. However, an engaging world also needs to be founded on a spatially coherent layout that consists of diverse landuse areas arranged rationally. For example, a natural scene is likely to have a residential area, a lake, and a forest, and the residential area is better to be in front of the lake and partially surrounded by the forest. With a rational layout, objects can be organized faithfully to the real scenes, which improves the experience of exploration. A virtual world further needs to be creative where human design can be applied to the layouts. For example, generating a layout where the center should be a commercial area and the rest should be composed of 60% forest and 40% meadow. Designing layouts requires a combination of logical and artistic considerations and demands a high level of expertise and effort from humans. It is thus important to develop algorithms for generating landuse layouts, which is considered a challenging task.

Layout generation is extensively studied in the context of graphic design [20, 17, 15, 1, 36] and indoor scene synthesis [3, 21, 31, 29, 18, 4, 23, 6]. The proposed models generate arrangements of objects regarding their categories, locations, rotation angles, and scales. However, virtual worlds involve more complex arrangements, and optimal generation of their layouts is challenging and little studied. In procedural generation [5, 25, 39, 11], layouts are synthesized by noise sampling or heuristic models which focus on terrains and biomes, and have no consideration of human landuse areas. To generate plausible layouts of virtual worlds, a promising direction is to develop approaches driven by the real-world landuse distribution.

We consider that virtual world layouts can be referenced from the real world. Therefore, we build a dataset of landuse layouts which are collected and processed from OpenStreetMap[1]. A landuse layout is a 2-dimension grid map of blocks, where each block is labeled by landuse categories. We propose **L**and**u**se **T**rans**f**ormer (LuTF) which is a generative model of landuse layouts conditioned on spatial and semantic hints. We define the spatial hint as partially observed layouts and the semantic hint as compositions of landuse percentages. LuTF is based on a transformer network due to its high expressiveness and natural adaptation to the discrete landuse category. The network architecture includes two encoders and one decoder. The first encoder takes partial layouts as input and the second encoder takes landuse compositions as input. The encoders jointly form a latent control representation, which is then cross-attended by the decoder to generate a layout autoregressively from "top-left" to "bottom-right", following suc-

[1] OpenStreetMap is released with an open-content license, which allows free access to all underlying map data. https://www.openstreetmap.org/

cessful image synthesis work [10, 28, 37, 27]. At the training stage, a partial layout is produced by masking a fully observed layout, and a composition is computed as the ratios of the landuses within the masked part. We feed the model various partial layouts with the corresponding compositions to reconstruct the full layouts. A partial layout can be completely masked, and a composition can be set to an "unknown" value, which represent no hint. The model can learn an aggregated layout distribution under no hint/control circumstances. At the inference stage, LuTF can be used to sample coherent layouts with or without user input controls.

To improve the quality of the generated layouts, we devise a geometry objective function that enhances the learning of shapes. In detail, for each block on a layout map, the model is supervised to predict its distance and direction to the centroid of its belonging area, and the number of its neighbors with different landuses. These supervisions reveal the geometry of the layouts, allowing the model to perceive shapes and generate layouts with geometric priors. We further develop a planning objective function to address an issue with autoregressive predictions. These predictions are conditioned on previous ones, resulting in fewer restrictions at the beginning and more at the end. If the decoding deviates early on, it can cause the final generation spatially irrational and out of control. Our planning objective function guides the model to perceive progressive composition demands while decoding. This imposes regulations on the model outputs for better quality and more conformity.

To generate layouts of unlimited sizes for practical use, we propose an expansion approach that iteratively grows an existing layout on its borders. We use a window sliding on a border to capture part of the existing layout as a hint, which is fed into the model for completion. This expansion approach can produce locally rational layouts without further training and modification to the model.

To evaluate the performance of the model, we propose two metrics. One is the Wasserstein distance which measures visual or spatial similarity between a set of layout generations and the real samples. We train a convolutional autoencoder that maps 2D layouts to vectors, and calculate the distance between the two vector distributions. The other metric is the difference between the compositions of generated layouts and the input composition, which measures the semantic conformity to the control. Finally, given a generated layout, terrains can be created and assets can be populated depending on the landuse areas to form a virtual world. An overview of our method is shown in Figure 1.

Contributions

- We formulate a problem of generating virtual world layouts with landuse distribution from the real world. To the best of our knowledge, this is the first work to study the problem.
- We propose a controllable generative model of layouts: **L**and**u**se **T**rans**f**ormer (LuTF), which encodes given spatial and semantic hints, and then decodes layouts of 2D landuse maps.
- To improve layout quality and conformity to the control, we design a geometry and a planning objective which supervise the model to perceive layout shapes and composition demands in progress. LuTF is therefore able to regularize generations with geometric priors and drive the outputs to meet the requirements.
- For evaluating generation quality from a spatial similarity perspective, we train an autoencoder to embed layouts into vectors, so that the generated and real-world layouts can be compared by the Wasserstein metric.

2. Related Work

Procedural Generation Procedural generation aims to automatically generate realistic content for virtual worlds such as landscapes, vegetation, road networks, buildings, and living beings [12]. The placement of the objects is usually based on noise sampling [25] or a Voronoi diagram [9] with considerations of geography and climate factors such as flatness, altitude, and humidity [11]. However, the procedural layouts are short of interesting human landuse areas and are heuristic compared to real-world distribution. It is thus important to investigate data-driven methods for generating optimal layouts.

Layout Generation in Graphic Design Layout generation is an important problem abstracted from designing images, documents, and software user interfaces. The study aims to automatically arrange a meaningful composition of graphic primitives. LayoutGAN [20] synthesizes bounding box annotations of 2D graphic elements based on semantic and geometric properties using a generative adversarial network (GAN) framework. LayoutVAE [17] takes a label set description to generate a full image layout by using an autoregressive model based on a conditional variational autoencoder (VAE) framework. LayoutTransformer [15] applies self-attention to learn contextual relationships between graphic elements, and enables generating a layout from an empty set of elements or an initial seed set of primitives. Variational Transformer Network (VTN) [1] captures high-level relationships between elements with self-attention layers and models the layout synthesis with the VAE framework. VTN is capable of learning margins, alignments, and other global design rules without explicit supervision. LayoutTransformer Network (LT-Net) [36] uniquely encodes the semantic features of scene graph elements for exploiting co-occurrences and implicit relationships. LT-Net can

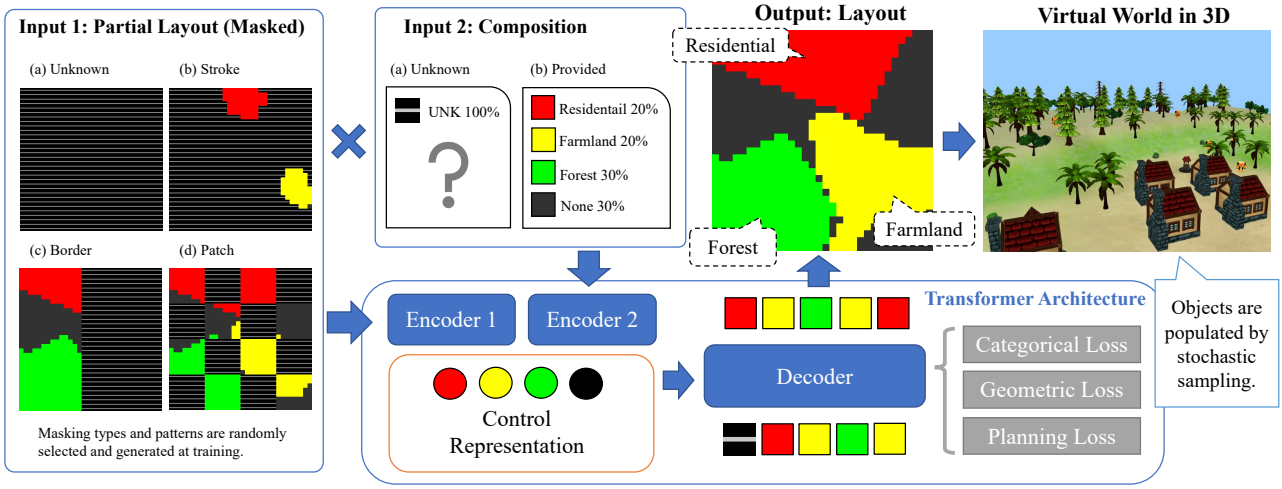

Figure 1: An overview of the method based on **L**and**u**se **T**rans**f**ormer (LuTF).

additionally produce a spatially-diverse layout by fitting a Gaussian mixture model on the bounding box distribution.

Indoor Scene Synthesis There is a large body of work on indoor scene synthesis, which aims to generate optimal arrangements of a set of indoor objects. Make It Home [38] starts from a randomly initialized layout, and iteratively adjusts it by minimizing a cost function regarding factors such as human-accessibility, visibility, pairwise object relationships, etc. PlanIt [31] generates indoor layouts with a deep graph convolutional model which synthesizes relation graphs of objects in an autoregressive manner. GRAINS [21] uses a recursive VAE based model to generate a scene structure of objects hierarchically from a random code. SceneGraphNet [40] takes an incomplete scene as context to predict the most likely object type given a query location. It uses a graph neural network to model long and short object relationships with a neural message passing approach. SceneGen [18] takes a semantic segmentation of an incomplete scene to predict a placement probability map for new objects. 3D-SLN [23] uses an end-to-end variational generative model to generate diverse layouts given scene graphs. A convolutional neural network based method [32] uses top-down images of scenes to predict the next object to insert and its location simultaneously. This method is later extended with factorizing the process which enables global reasoning of adding an object and considers more geometric dimensions like scales and rotations [29]. For more flexible scene synthesis, natural language is used to control the layout generation. Early work [2, 3] parses text input into a set of object constraints, then expands the set by inference and arranges objects. A novel framework [24] considers scene synthesis at a sub-scene level which regularizes the outputs. It parses textual commands into a semantic scene graph to retrieve sub-scenes from a database, then the final scene is synthesized by aligning the sub-scenes with augmentation of adding new objects. The House Plan Generative Model [4] translates text into a structural graph representation, and predicts the layout by a graph convolutional network and generates interior texture by a language conditioned generative adversarial network.

Other Layouts with Deep Generative Models Compared to graphic design and interior scene synthesis, layouts of virtual worlds arrange a lot more elements with complex and implicit relations, which are intractable to be represented by sparse annotations and modeled using the above methods. Generating virtual world layouts therefore requires designing models with high expressiveness and powerful generative capability.

A considerable amount of effort has been devoted to generating other types of layouts by deep generative models [19, 13, 26]. BlockPlanner [35] generates large-scale city blocks. It uses a vectorized graph representation for land lots which enables a lightweight graph VAE to capture the hidden distributions. 3D shape structures can be considered as layouts as well. SAGNet [34] is a structure-aware generative model for 3D shapes. It embeds the geometry of object parts and the pairwise relationships jointly in a latent space, that is learned by an autoencoder. MatFormer [14] generates procedural material graphs which are viewed as compact, parametric, and resolution dependent spatial layouts for material authoring. It addresses the problem with a model of multi-stage transformers which sequentially generates nodes and edges while ensuring semantic validity.

Transformers with powerful modeling architecture for

long-range relations, have achieved state-of-the-art performance on various learning tasks [30, 8, 22]. Even for spatially distributed images, the transformer architecture contains no built-in inductive prior on the locality of interactions compared with vision predominant convolutional neural nets, and is therefore free to learn complex relationships among its inputs. A lot of work has demonstrated the success of using transformers to synthesize natural images [10, 28, 37, 27]. Transformers have also been investigated for scene layout generations. Sceneformer [33] predicts a sequence of objects along with their locations and orientations. It improves quality and shows faster generation compared to previous methods. The model is also flexible to be conditioned on text descriptions due to modality compatibility. Inspired by those recent successes, we study the virtual world layout generation based on transformer architecture and augment it with novel designs.

3. Data Description

As virtual world layouts can be referenced from the real-world distribution, we use OpenStreetMap which is a free geographic database of the world. We take the data of "Greater London" from OpenStreetMap. The data contains a vast number of lands labeled by landuse which describes primary human usage. For example, a land can be labeled as forest, meadow, farmland, etc. As there are dozens of landuse categories, and most of them are extremely rare, we only consider those types with a coverage ratio of more than 1%: "residential", "farmland", "industrial", "meadow", "grass", "retail", "recreation ground", "forest", "commercial", "railway", and "cemetery". The coverage details are presented in Table 1. For the areas not covered by any land, we label them as "none" so that there are 12 categories in total. Each land is represented by a polygon where vertices are coordinates. We use a sliding window of size 512 $\times$ 512 meters with a stride of 256 meters to extract sample regions from the entire land plane. Each sample captures a number of land polygons (cropped if one intersects with the window). We exclude the sampled regions which are fully covered by one landuse category as they provide no information for learning layout organizations. The final number of samples is 21864. Following state-of-the-art image generation approaches [28, 10], we divide the sampled regions into $L^2 = 32 \times 32$ grids of equal-sized blocks, and each block is labeled as the landuse of the largest covering polygon. Therefore, a sampled region is converted into a 2-dimensional categorical map of blocks to represent a landuse layout. We show a few samples in Figure 2.

4. Methodology

In this section, we first introduce primary notations used in the paper. We then discuss our method of **L**and**u**se **T**rans**f**ormer (LuTF) with the model architecture and learning objectives. Finally, we describe the training and inference procedures of our model, and an approach to expand layouts for practical use.

4.1. Notations

We denote a layout map as x. We denote a partial layout as $\dot{x}$ where a number of blocks are labeled as a mask value. We denote a landuse composition as c which states the area percentage of different landuse categories. Let M be the total number of landuse categories, c can be formally presented as a sequence of pairs $\{c_i = (u_i, r_i) \,|\, i = 1, ..., M\}$ where u_i is a one-hot vector indicating the category, and r_i is a scalar of the area percentage that $\sum_{i=1}^{M} r_i = 1$.

4.2. Landuse Transformer (LuTF) Model

4.2.1 Model Architecture & Basic Objective

Let $D = \{x\}$ be a layout dataset. We mask x to produce $\dot{x}$ as a spatial hint. $\dot{x}$ can be fully masked as "unknown" for no spatial control. We compute c for those masked blocks of x as a semantic hint. c can also be an "unknown" value for no semantic control. Our goal is to learn a conditional generative model $p(x|\dot{x}, c)$. We propose LuTF which utilizes highly expressive transformer architecture to model x. We reshape a 2-dimensional x to a block sequence $\{x_t \,|\, t = 1, ..., T\}$ of length $T = L^2$ using "top-left" to "bottom-right" order, and each block x_t can be viewed as a discrete token for the transformer. The architecture of LuTF includes two encoders and one decoder, each one is composed of a stack of transformer layers. The first encoder takes a partial layout $\dot{x}$ to generate a sequential representation h_1. The second encoder takes a composition c to generate a representation h_2. The final latent control representation h is concatenated as:

$$h = h_1 \oplus h_2 = \text{Encoder}_1(\dot{x}) \oplus \text{Encoder}_2(c). \quad (1)$$

Let $x_{<t} = \{x_1, x_2, ..., x_{t-1}\}$. The decoder conditioned on h to generate x via a probabilistic model $p(x|h)$ which can be factorized as:

$$p(x|h) = \prod_{t=1}^{T} p(x_t|x_{<t}, h). \quad (2)$$

We therefore formulate $p(x|h)$ as an autoregressive model. In the decoder self-attention modules, token values only attend to the previous ones. In the cross-attention modules, h induces keys and values which can be considered as a landuse platter. The t-th query token, which encodes the information of $x_{<t}$, checks the platter and finds a proper representation for the t-th position from the platter. The last layer of the decoder generates hidden states z, and a linear layer $f(z)$ transforms z into an M-dimensional vector for

Landuse	Area	Coverage	Color	Landuse	Area	Coverage	Color
residential	775.10 km^2	64.25%		recreation ground	30.51 km^2	2.53%	
farmland	148.82 km^2	12.34%		forest	20.54 km^2	1.70%	
industrial	51.38 km^2	4.26%		commercial	14.52 km^2	1.20%	
meadow	38.38 km^2	3.18%		railway	14.44 km^2	1.20%	
grass	31.56 km^2	2.62%		cemetery	13.02 km^2	1.08%	
retail	30.56 km^2	2.53%		none	-	-	

Table 1: The landuse coverage in the "Greater London" region.

Figure 2: Samples of landuse layouts in the "Greater London" region.

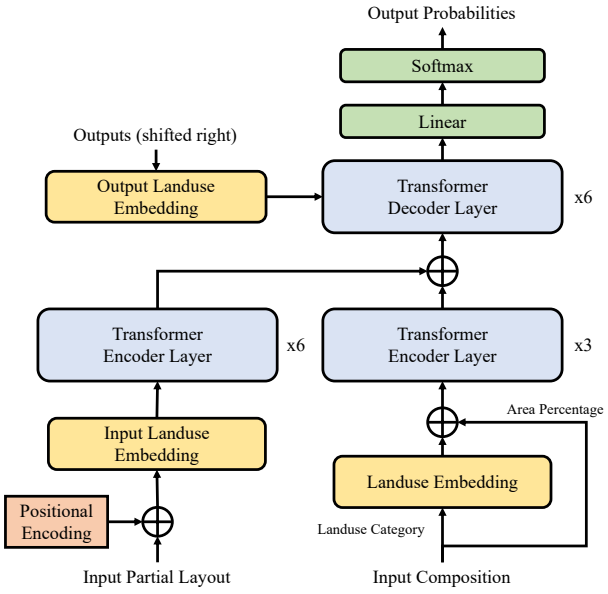

Figure 3: The model architecture of Landuse Transformer (LuTF).

computing $p(x_t|x_{<t}, h)$ = softmax$(f(z))$. The end-to-end transformer model is expressed as:

$$p(x_t|x_{<t}, \dot{x}, c) = \text{Decoder}(x_{<t}, \text{Encoder}_1(\dot{x}), \text{Encoder}_2(c)), \tag{3}$$

which can be learned by minimizing the objective of negative log-likelihood loss regarding landuse category:

$$\mathcal{L}_{\text{landuse}} = \sum_{x_t \in x} -\log p(x_t|x_{<t}, \dot{x}, c). \tag{4}$$

The architecture of LuTF model is depicted in Figure 3.

4.2.2 Geometry & Planning Objectives

We improve the layout quality by introducing two additional objectives for training LuTF. We propose a geometry objective function to learn geometric properties, that the model is capable of perceiving shapes and generating layouts with shape priors. We devise the supervision at the block level. For each block, the model is expected to predict three geometric descriptors: 1) the distance to the centroid of the area to which the block belongs; 2) the direction from the block to the centroid; 3) the number of neighbors with different landuse categories. The distances, directions, and neighbor counts describe the shape of an area. For example, a "ring" area has many equally distanced blocks, a "stick" area majorly has two opposite directions, and an area of complex shape has more neighbors than a simplex area. Given a layout x, we compute the three descriptor maps: d for distances, θ for directions, and b for neighbor counts. Let vec(t) be the direction vector of the spatial coordinate at t to its corresponding centroid, and neb(t) be the adjacent neighbor set at t. The three descriptor maps

are defined as:

$$d = \{d_t = \frac{1}{\sqrt{2}L}||\text{vec}(t)||_2 \mid t = 1, ..., T\}, \quad (5)$$

$$\theta = \{\theta_t = \frac{1}{2\pi}\arctan(\text{vec}(t)) \mid t = 1, ..., T\}, \quad (6)$$

$$b = \{b_t = \frac{1}{|\text{neb}(t)|}\sum_{\tau \in \text{neb}(t)} \mathbf{1}\{x_t \neq x_\tau\} \mid t = 1, ..., T\}. \quad (7)$$

We introduce three linear layers $f_d(z)$, $f_\theta(z)$, $f_b(z)$ on the decoder hidden states z. Let $\tilde{d}$, $\tilde{\theta}$, and $\tilde{b}$ be the model predictions of the three descriptors, where $\tilde{d}_t$ = sigmoid($f_d(z)$), $\tilde{\theta}_t$ = sigmoid($f_\theta(z)$), and $\tilde{b}_t$ = sigmoid($f_b(z)$). The geometry objective function is the sum of the l_1 distances between the predictions and the targets:

$$\mathcal{L}_d = ||\tilde{d} - d||_1, \quad (8)$$

$$\mathcal{L}_\theta = ||\tilde{\theta} - \theta||_1, \quad (9)$$

$$\mathcal{L}_b = ||\tilde{b} - b||_1, \quad (10)$$

which are minimized together with $\mathcal{L}_{\text{landuse}}$. We use the l_1 loss function to prevent the model fitting outliers of extremely irregular shapes.

Recall that we generate x with an autoregressive transformer decoder, where the t-th prediction is conditioned on the previous predictions. This causes the model generations to have fewer restrictions at the beginning and more at the end. Therefore, when the early generations largely deviate from the control, the final output can be irrational and not meet the requirements. For example, the model may generate a large "forest" so that there is no space for an expected "farmland" area. To impose control, we devise a planning objective that enables the model to perceive step-by-step composition demands while decoding. It guides the model to make rectifications on-the-fly, so that the final output can be more spatially rational and under control. Let $x_{\geq t}$ = $\{x_t$, x_{t+1}, ..., $x_T\}$, and $c_{\geq t}$ be the composition of $x_{\geq t}$ which is demanded at the t-th step. The model is expected to predict $c_{\geq t}$, which can be simplified as a vector:

$$c_{\geq t} = (r_{\geq t,1}, r_{\geq t,2}, ..., r_{\geq t,M}), \quad (11)$$

where $r_{\geq t,i}$ is the area percentage of the i-th landuse category. Let the model prediction be $\tilde{c}_{\geq t}$, and we define the planning loss as the sum of Kullback-Leibler divergences between $c_{\geq t}$ and $\tilde{c}_{\geq t}$ as:

$$\mathcal{L}_{\text{plan}} = \sum_{t=1}^{T}\sum_{i=1}^{M} r_{\geq t,i} \log \frac{r_{\geq t,i}}{\tilde{r}_{\geq t,i} + \epsilon}, \quad (12)$$

where ϵ is a constant to avoid division by zero.

In summary, with the awareness of this dynamic composition demand, the model can improve the generation conformity to the control. Combining the loss functions regarding landuse category, geometric descriptors, and planning, the final learning objective is expressed as:

$$\mathcal{L} = \sum_{(x,c)\in D} \mathcal{L}_{\text{landuse}} + \lambda_1\mathcal{L}_d + \lambda_2\mathcal{L}_\theta + \lambda_3\mathcal{L}_b + \lambda_4\mathcal{L}_{\text{plan}}, \quad (13)$$

where λ_1, λ_2, λ_3, and λ_4 are the hyper-parameters for weighting each loss.

4.3. Training & Inference

The training of LuTF requires producing a partial layout $\dot{x}$ and a composition c for each x. We introduce four types of partial layout regarding masking strategy: 1) Unknown: completely masked; 2) Stroke: a few stroked parts are visible, and the rest are masked; 3) Border: a layout is vertically or horizontally cut into halves, and one half is masked; 4) Patch: a layout is divided into grids of patches, and some of them are masked. While iterating the dataset, we randomly choose one of the masking types to produce $\dot{x}$, then c can be either computed on the masked part $x - \dot{x}$ or set to "unknown". To enable autoregressive decoding, we put an x_0 labeled as "start" at the beginning of x. Then, the model input is a map x' = $\{x_0, x_1, ..., x_{T-1}\}$ along with $\dot{x}$ and c, and the model output is supervised by the original x = $\{x_1, x_2, ..., x_T\}$ with d, θ, b, and all $c_{\geq t}$.

The inference of LuTF is based on causal sampling. A user can provide a partial layout $\dot{x}$ and an arbitrary composition c. They are set to "unknown" if not provided. We feed them into the encoders to obtain the control representation h. We start with x_0, and use the decoder to progressively generate the next hidden state $z_t = z$. We perform top-K filtering on values of $f(z)$, i.e, only the K largest values are kept and others are set to negative infinity. The probability $p(x_t|x_{<t}, h)$ is computed as softmax($f(z)$) where we draw out x_t using the multinomial sampling. We iterate this procedure for T steps and obtain the final inference.

4.4. Layout Expansion

To generate large layouts for virtual worlds, we additionally present an expansion approach based on the LuTF model. Given an existing layout, we can extend one of its borders, for example, the upper border. We place an $L \times L$ window at the border with the bottom half overlapping the existing layout, and produce an $\dot{x}$ with the top half set to mask values. The model predicts the landuse of the top half which is then stitched to the layout. If the border length is longer than L, we slide the window with overlaps of newly inferred regions, for maintaining the continuity of the generations between steps. We illustrate the expansion procedure in Figure 4. With this approach, an existing layout can be expanded to an unlimited size. It might be a concern that

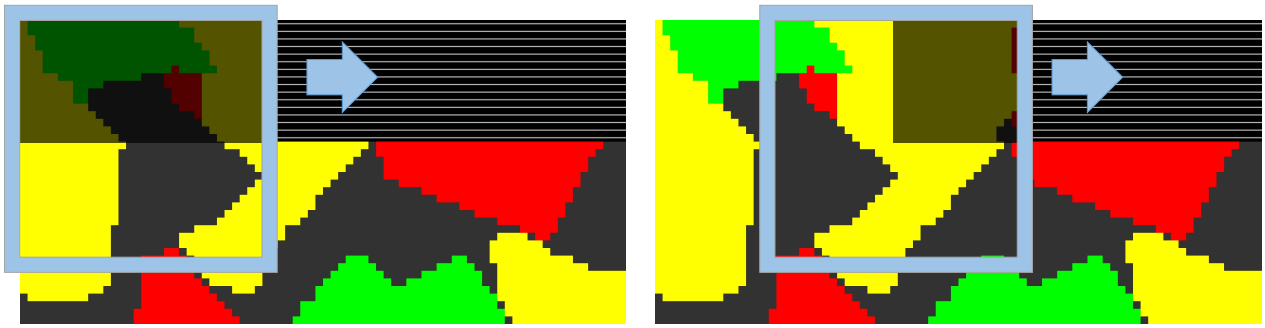

Figure 4: A window slides along the border, which captures part of existing layout as spatial hint for inference. The dark part in the window is the generated partial layout, which is then stitched to the border for expansion. The strips outside the window represent the masked parts waiting to be completed in the future.

the entire expanded layout is not optimized regarding spatial distribution. However, since a single $L \times L$ region can be very large for users to perceive (above 1/4 km^2 in this study), we consider that ensuring coherence on sub-regions would be enough for virtual world layouts in general.

5. Evaluation Metrics

To the best of our knowledge, there is no existing work on landuse layout generation for virtual worlds. We therefore propose two metrics for performance evaluation. The first one is the Wasserstein metric inspired by Fréchet inception distance [16]. We train a convolutional autoencoder that converts layout maps into encoding vectors. Suppose $\mathcal{N}(\mu, \Sigma)$ is the distribution of encoding vectors of the generated layouts, and $\mathcal{N}(\mu', \Sigma')$ is the distribution of encoding vectors of the real-world layouts. The Wasserstein metric m_w is computed as:

$$m_w = ||\mu - \mu'||_2^2 + \text{tr}(\Sigma + \Sigma' - 2(\Sigma\Sigma')^{\frac{1}{2}}), \tag{14}$$

where tr is the trace of a matrix. This metric indicates how generated layouts are spatially similar to the real-world. The second metric is the difference between the provided composition c and $\tilde{c}$ of generated $\tilde{x}$. We compute this metric denoted by m_c as:

$$m_c = \frac{1}{2M} \sum_{i=1}^{M} |r_i - \tilde{r}_i|. \tag{15}$$

If c is equal to $\tilde{c}$, then $m_c = 0.0$. If c is completely different to $\tilde{c}$, for example, $c = (0.4, 0.0, 0.6)$ and $\tilde{c} = (0.0, 1.0, 0.0)$, then $m_c = 1.0$. This metric indicates how the composition of a layout fits to user control. We consider that a high-quality and more satisfactory generation should be with both small m_w and m_c.

6. Experiments

6.1. Implementation Details

The landuse categories used in the experiments are listed in Table 1. We also add a category of "UNK" which is used as mask value and as the start sign of the decoder. There are $M = 13$ landuse categories in total. The model implementation is based on PyTorch 1.8. The transformer configuration of the proposed model is as follows: The partial layout encoder has 6 transformer layers, and the composition encoder has 3 transformer layers, where the hidden dimension is 64, the head number is 8; The decoder has 6 transformer layers, where the hidden dimension is 64, the head number is 8, and with an upper triangle attention mask. The autoencoder proposed to evaluate the m_w metric consists of an embedding layer, 3 convolution layers, and 4 devconvolution layers, and the encoding vector is 256-dim normalized into [0, 1]. We train and evaluate the methods on a Tesla V100 GPU.

6.2. Comparison Methods & Settings

We setup the comparison methods as follows:

- Voronoi: The classic method for game world layout generation using the Voronoi diagram. Given a composition c, the number of Voronoi cells is set to the count of unique landuses which have a ratio greater than zero. If c is not given, the cell number is chosen randomly within $[1, M]$ (M is the total number of landuse categories). The site points are randomly placed on the layout for generating cells. Note that there is no learning procedure.
- CVAE: The model is based on a conditional variational autoencoder with convolutional architecture. The encoder learns a normal distribution from a joint representation of layout x, partial layout $\dot{x}$, and composition

Method (Eval m_w)	U-U	U-M	S-U	S-M	B-U	B-M
Voronoi	7.7848	2.9410	5.9547	1.6883	2.8773	0.6596
CVAE	16.1266	15.3241	4.7707	3.6969	2.0580	1.6160
CGAN	15.0759	7.0437	0.6624	0.5994	0.4498	0.4146
LuTF	0.6854	0.2776	0.1398	0.1339	0.1060	0.0948
LuTF + G	0.7030	0.2469	0.1473	0.1320	0.1075	0.0900
LuTF + G + P	**0.6745**	**0.2239**	**0.1393**	**0.1267**	**0.1030**	**0.0870**
Method (Eval m_c)	U-U	U-M	S-U	S-M	B-U	B-M
Voronoi	-	44.90 (17.55)	-	32.94 (14.87)	-	22.24 (17.93)
CVAE	-	50.70 (22.73)	-	37.64 (18.61)	-	24.84 (20.39)
CGAN	-	28.63 (14.31)	-	9.97 (7.86)	-	8.72 (9.16)
LuTF	-	15.32 (11.97)	-	9.32 (7.69)	-	7.66 (8.36)
LuTF + G	-	9.31 (8.27)	-	6.96 (5.71)	-	5.73 (6.08)
LuTF + G + P	-	**8.70** (7.56)	-	**6.96** (5.66)	-	**5.69** (5.93)

Table 2: The top half shows the m_w values (the Wasserstein distances). The bottom half shows the m_c values with standard deviations in the brackets, where the numbers are in 100%.

Method	U-R	S-R	B-R	U-R	S-R	B-R
	m_w			m_c		
Voronoi	8.1185	5.6584	2.7576	**29.04** (13.81)	**21.76** (11.25)	**16.40** (12.80)
CVAE	15.7308	4.2976	1.9934	83.13 (17.34)	60.88 (17.18)	41.09 (27.30)
CGAN	9.7293	3.0658	1.9797	37.07 (15.65)	29.41 (12.09)	17.91 (13.64)
LuTF	8.6048	**2.4484**	1.6374	46.73 (21.33)	39.77 (14.65)	25.76 (18.51)
LuTF + G	8.0324	2.5395	**1.5790**	40.60 (20.73)	37.39 (14.96)	24.18 (17.29)
LuTF + G + P	**7.7088**	2.6335	1.6202	39.05 (20.54)	35.01 (14.50)	22.86 (16.77)

Table 3: The results of using Random compositions. The left half shows the m_w values. The right half shows the m_c values with standard deviations in the brackets, where the numbers are in 100%.

c. The decoder generates a layout using a sampled latent vector with the conditions. The CVAE is trained with reconstruction loss.

- CGAN: The model is based on a conditional generative adversarial network with convolutional architecture. The generator composes an encoder that maps $\dot{x}$ and c with random noise to a latent vector, and a decoder that decodes the vector to x. The discriminator judges whether a generation is real or fake. The CGAN is trained with both reconstruction and adversarial loss.

- LuTF: The proposed transformer model with only categorical loss.

- LuTF + G: The proposed model with categorical + geometry (G) loss.

- LuTF + G + P: The proposed model with categorical + geometry (G) + planning (P) loss.

We randomly split the dataset into three parts: 70% for training, 10% for validation, and 20% for testing. We train the models for 1000 epochs using the Adam optimizer with a learning rate of 0.0001 and a batch size of 16. The hyperparameters λ_1, λ_2, λ_3, and λ_4 are all set to 0.1. The model inference uses top-5 filtering. The autoencoder is trained for 70 epochs using the Adam optimizer with a learning rate of 0.001 and a batch size of 16.

We evaluate the methods with combinations of various spatial and semantic inputs. We produce 3 types of partial layout input $\dot{x}$: Unknown (U), Stroke (S), and Border (B), which have been described in section 4.3. We produce 2 types of composition input c: 1) Unknown (U): the ratio of "UNK" is 100% ; 2) Masked (M): c is computed on the masked part of a layout sample. For simplicity, we denote a combination of inputs with the abbreviate letters in the brackets. For example, an $\dot{x}$ of Stroke (S) with a c of Unknown (U) is denoted as S-U. Totally, we use 6 different combinations of inputs to evaluate the methods.

Figure 5: A real-world layout with three partial layout inputs $\dot{x}$: Unknown (U), Stroke (S), and Border (B).

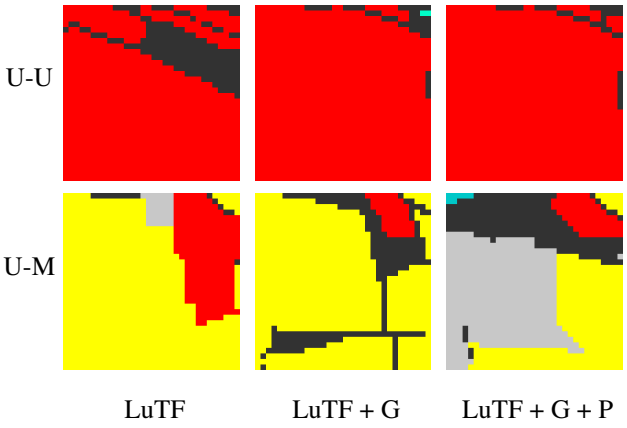

Figure 6: The 1st row is the results of input U-U: Unknown (U) partial layout $\dot{x}$ and Unknown (U) composition c. The 2nd row is the results of U-M: Unknown (U) $\dot{x}$ and Masked (M) c.

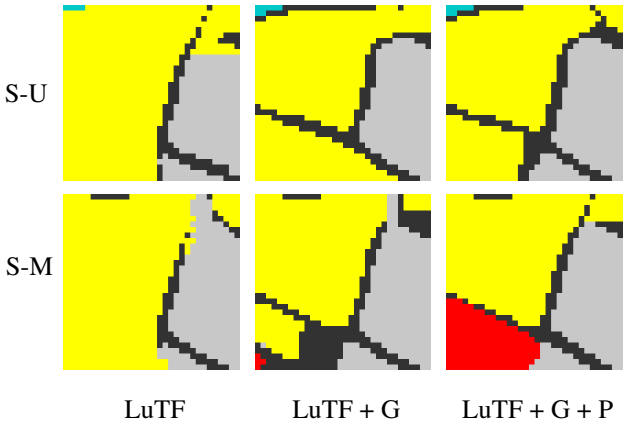

Figure 7: The 1st row is the results of input S-U: Stroke (S) partial layout $\dot{x}$ and Unknown (U) composition c. The 2nd row is the results of S-M: Stroke (S) $\dot{x}$ and Masked (M) c.

6.3. Primary Results & Analysis

The experimental results regarding m_w and m_c are shown in Table 2. Note that for those combinations with Unknown c (U-U, S-U, B-U), evaluations regarding m_c are omitted as there is no composition for comparison. The results show that all the LuTF based methods significantly outperform Voronoi, CVAE, and CGAN due to the strong generation capability of the transformer architecture. With geometry and planning objective functions, LuTF + G + P outperforms the original LuTF on m_w and m_c, which demonstrates the advantages of the proposed supervisions to improve layout quality and conformity. LuTF + G outperforms LuTF on m_w and m_c for most of the evaluation input types. As the geometry objective function facilitates the model to perceive shapes, the generated layouts are more coherent and organized in geometry, which leads to the improvement on m_c. We notice that there is a slight dropping of spatial realness (higher m_w) for those experiments without composition control (Unknown c: U-U, S-U, B-U). As LuTF + G leans to produce layouts which are more diverse in terms of shapes, the spatial distance to the real-world layouts could be increased. When a composition control is given, LuTF + G achieves lower m_w since it can generate more complex shapes that meet the control. As a result, the geometry objective generally improves spatial coherence and conformity. LuTF + G + P outperforms LuTF + G on both m_w and m_c for all the input types. As the planning objective injects the model a semantic preview of generations, decoding deviations at early stages can be suppressed. Therefore, the planning objective also improves spatial rationality and conformity to the control.

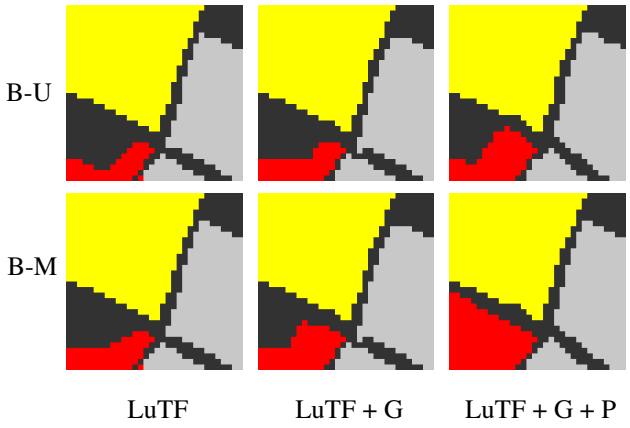

Figure 8: The 1st row is the results of input B-U: Border (B) $\dot{x}$ and Unknown (U) c. The 2nd row is the results of B-M: Border (B) $\dot{x}$ and Masked (M) c.

We use a layout sample to qualitatively analyze LuTF based methods and demonstrate the advantages of the proposed objectives. We choose a real-world layout, and produce three partial layout inputs $\dot{x}$ of Unknown, Stroke, and Border, which are shown in Figure 5. Given the Unknown $\dot{x}$, we show the results of inputs U-U and U-M in Figure 6. For U-U, all the methods generate reasonably simple layouts as there is no control. For U-M, LuTF and LuTF + G

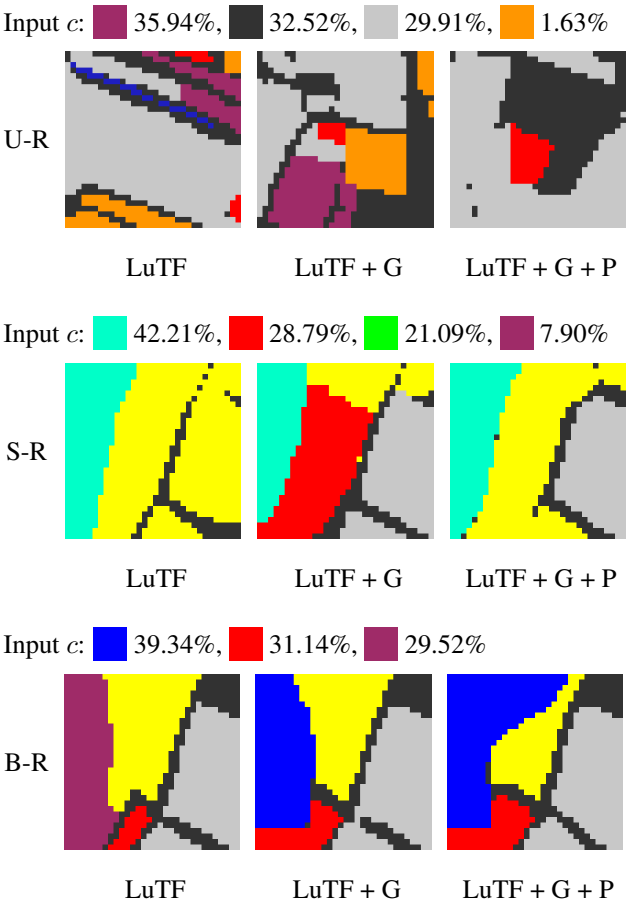

Figure 9: The results of Unknown (U), Stroke (S), Border (B) $\dot{x}$ with Random (R) composition c. The landuse ratios of c are shown above each row. Note that the input c controls the generation of the masked part of the input $\dot{x}$, shown in Figure 5.

fail to generate a large enough "industrial" ( ) area, while LuTF + G + P generates a more satisfied layout given the control of composition, due to the advantage of the planning objective. Given the Stroke $\dot{x}$, we show the results of inputs S-U and S-M in Figure 7. For S-U, both LuTF + G and LuTF + G + P recover a left-to-right "road" of "none" ( ) inferred from the input partial layout, due to the advantage of the geometry objective. For S-M, LuTF + G + P generates and places a "residential" ( ) area properly. Given the Border $\dot{x}$, we show the results of inputs B-U and B-M in Figure 8. Overall, all the methods perform well, while LuTF + G + P generates spatially more coherent layouts regarding the "residential" ( ) area.

6.4. Random Composition

A virtual world also needs to be creative so that users can apply various designs to the layouts. Therefore, we evaluate the methods with randomly generated composition inputs simulating user creative ideas. For each masked layout in the testing data, we first obtain the number n of unique landuse in the masked part, and produce c with random n landuses where random ratios are assigned (normalized to 1). We denote this type of input c as Random (R). Generating layouts with Random c is challenging, as its distribution is different from the real world, which makes the model hard to maintain both spatial rationality and conformity.

The evaluation results regarding m_w and m_c are shown in Table 3. In terms of m_w, LuTF based methods all outperform Voronoi, CVAE, and CGAN, which demonstrates better spatial generalization ability. LuTF with geometry objective shows lower m_w for U-R and B-R but higher for S-R. The reason might be that, adopting shape priors can complete complex shapes between strokes, but the outputs can be less similar to the real-world layouts. In terms of m_c, Voronoi naturally outperforms the others as it forces to generate n cells of the required landuses in c without considering spatial rationality. CGAN performs as the second best, due to that the adversarial learning is capable of discriminating "fake" generations unfaithful to the input c. LuTF + G + P performs closely behind CGAN, but outperforms LuTF + G and LuTF, due to that the planning objective makes better control of the decodings. In general, the geometry and planning objective functions jointly improve spatial rationality and conformity.

We qualitatively study the results using the same layout sample shown in Figure 5. For U-R, LuTF + G generates the most satisfied layout that recovers all the required landuses but with a bit discrepancy in the ratios. For S-R, all the methods recover "grass" ( ) while only LuTF + G recovers "residential" ( ). For B-R, LuTF recovers "commercial" ( ), while LuTF + G and LuTF + G + P recover "recreation ground" ( ). As spatial rationality is considered, generating layouts of irregular landuse compositions can be suppressed as the models tend to maintain similarity to the real-world layouts.

6.5. Layout Expansion

We evaluate the quality of expanded layouts produced by the proposed expansion approach described in section 4.4. Recall that we train the model with samples of size 32×32 corresponding to 512×512 meters regions. We test the model with 3 groups of larger-sized layout samples: 64×64 (4x), 96×96 (9x), and 128×128 (16x). Given a testing sample, we crop out its central layout of size 32×32 as a starting point. We use the border expansion approach to extend its size spirally until reaches the original size, where the composition of the testing sample is used as control. The extended layouts are compared with the original layouts in terms of m_w and m_c. The results are shown in Table 4. LuTF + G + P outperforms the others in all the evaluations,

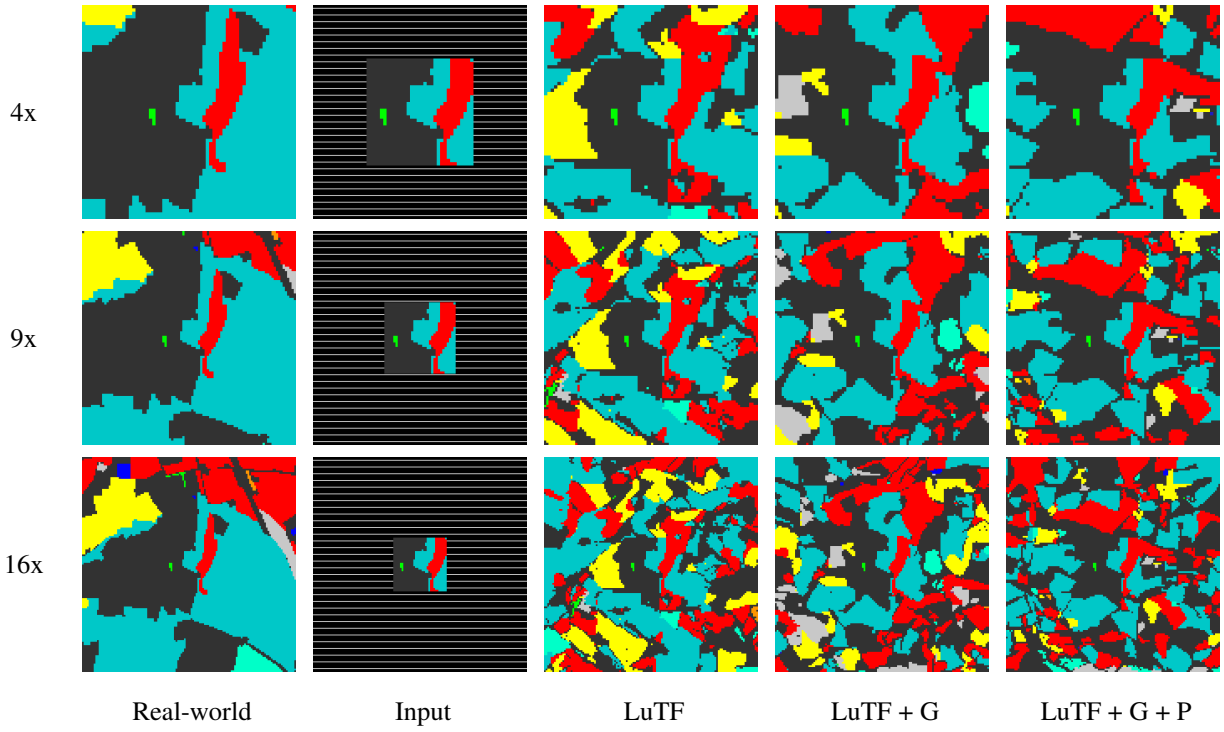

Figure 10: The figure shows expanded layouts of sizes 4x, 9x, and 16x. The images are scaled to the same size for a clean presentation.

Scale	4x	9x	16x
	m_w		
LuTF	4.2960	4.8279	5.1633
LuTF + G	4.2621	**4.6278**	5.0536
LuTF + G + P	**4.2367**	4.7194	**4.9176**
	m_c		
LuTF	7.76 (4.64)	7.70 (4.43)	7.93 (4.41)
LuTF + G	6.23 (4.09)	6.54 (3.99)	6.03 (3.41)
LuTF + G + P	**6.00** (3.11)	**6.31** (3.90)	**5.29** (3.13)

Table 4: The results of m_w and m_c for the expanded layouts.

except m_w at the 9x scale.

We show a few examples to visualize the quality of the expanded layouts. As shown in Figure 10, maintaining spatial coherence can be challenging with increased scales. Although the layouts are visually scattered, an ordinary user inside a virtual world can hardly perceive the entire 2048×2048 meters region. If global coherence has to be ensured, a multi-scale learning strategy can be applied. But there are always larger sizes that cannot be covered, which leaves an open question to study in the future.

6.6. Generating Virtual Worlds

Layouts provide important guidance for generating terrains and placing 3D objects in virtual worlds such as buildings, plants, and animals. In the experiment, we generate virtual worlds based on the layouts from the proposed LuTF. We first use Perlin noise sampling [25] to generate terrains, and scale the undulations depending on the landuse. For example, "residential" is usually on flat areas which needs a small scaling factor. For each area in the layout, we populate objects depending on the landuse. For example, we populate houses in "residential" areas, and plants in "meadow" areas. We build a system using ThreeJS framework[2] to synthesis and render virtual worlds. We show a few layouts and rendered views of the corresponding virtual worlds in Figure 11, 12, and 13.

7. Conclusions

In this paper, we study the problem of generating landuse layouts for virtual worlds by learning real-world geographic data. We propose Landuse Transformer (LuTF), a controllable generative model based on transformer architecture. The model takes inputs of spatial and semantic hints, and causally outputs a 2D map of landuse. The model can be used to sample diverse layouts under user controls, and

[2]ThreeJS: https://threejs.org/

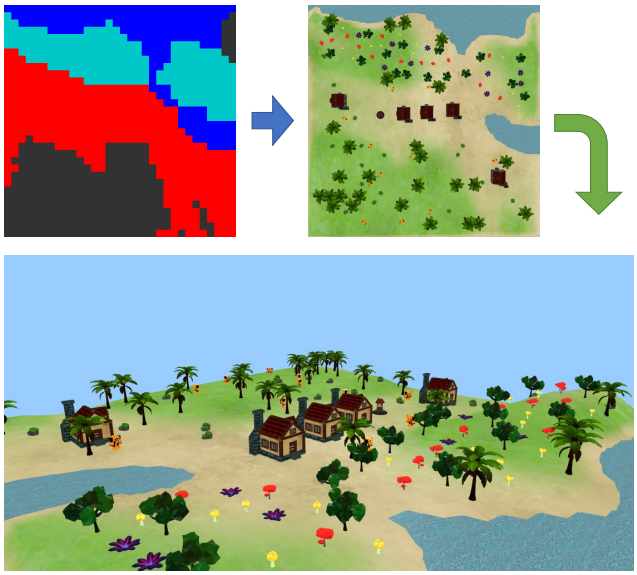

Figure 11: A layout, an aerial view of the generated 3D virtual world, and a view inside the world.

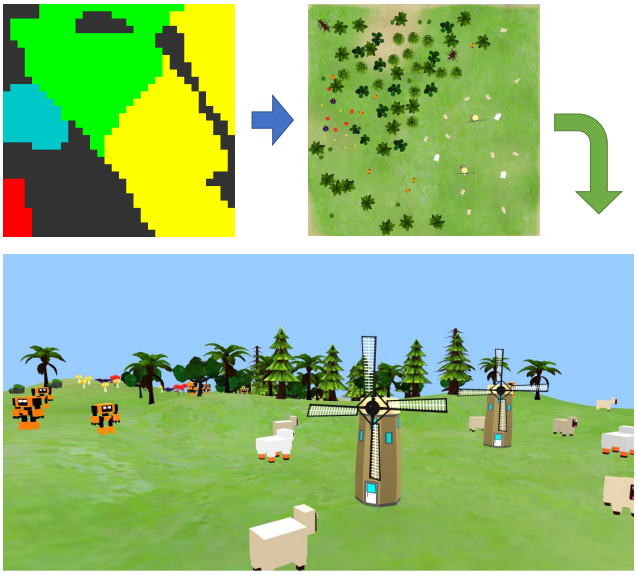

Figure 12: A layout, an aerial view of the generated 3D virtual world, and a view inside the world.

scale up existing layouts to an unlimited size. To generate high-quality and satisfactory layouts, we integrate two objective functions: one that supervises the model to perceive layout shapes and enhances the generations with geometric priors; the other supervises the model to perceive step-by-step composition demands and suppress the generations deviating from controls. To evaluate at the spatial level, we

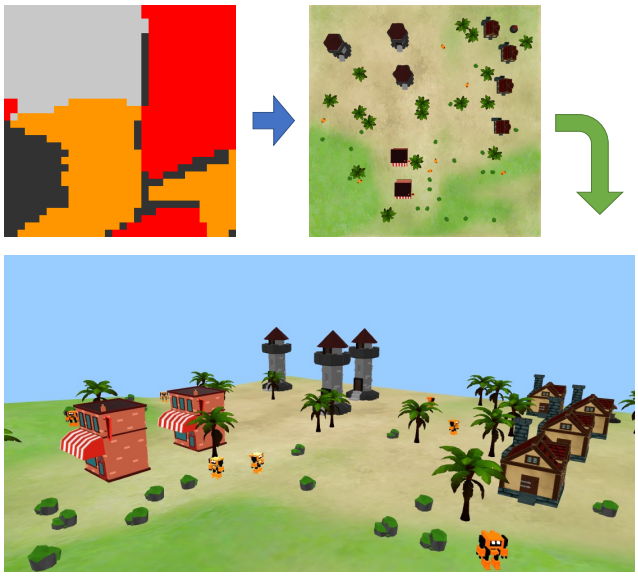

Figure 13: A layout, an aerial view of the generated 3D virtual world, and a view inside the world.

train an autoencoder to embed landuse layouts into vectors so that real and generated data can be compared using the Wasserstein distance. Given a generated layout, objects can be populated using stochastic sampling within each landuse area to form a virtual world.

References

[1] D. M. Arroyo, J. Postels, and F. Tombari. Variational transformer networks for layout generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13642–13652, 2021. 1, 2

[2] A. Chang, M. Savva, and C. D. Manning. Learning spatial knowledge for text to 3d scene generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2028–2038, 2014. 3

[3] A. X. Chang, M. Eric, M. Savva, and C. D. Manning. Sceneseer: 3d scene design with natural language. *arXiv preprint arXiv:1703.00050*, 2017. 1, 3

[4] Q. Chen, Q. Wu, R. Tang, Y. Wang, S. Wang, and M. Tan. Intelligent home 3d: Automatic 3d-house design from linguistic descriptions only. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12625–12634, 2020. 1, 3

[5] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 275–286, 1998. 1

[6] H. Dhamo, F. Manhardt, N. Navab, and F. Tombari. Graph-to-3d: End-to-end generation and manipulation of 3d scenes using scene graphs. In *Proceedings of the IEEE/CVF In-*

ternational Conference on Computer Vision, pages 16352–16361, 2021. 1

[7] J. D. N. Dionisio, W. G. B. III, and R. Gilbert. 3d virtual worlds and the metaverse: Current status and future possibilities. *ACM Computing Surveys (CSUR)*, 45(3):1–38, 2013. 1

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 4

[9] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003. 2

[10] P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12873–12883, 2021. 2, 4

[11] R. Fischer, P. Dittmann, R. Weller, and G. Zachmann. Autobiomes: procedural generation of multi-biome landscapes. *The Visual Computer*, 36(10):2263–2272, 2020. 1, 2

[12] J. Freiknecht and W. Effelsberg. A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*, 1(4):27, 2017. 2

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 3

[14] P. Guerrero, M. Hašan, K. Sunkavalli, R. Měch, T. Boubekeur, and N. J. Mitra. Matformer: A generative model for procedural materials. *arXiv preprint arXiv:2207.01044*, 2022. 3

[15] K. Gupta, J. Lazarow, A. Achille, L. S. Davis, V. Mahadevan, and A. Shrivastava. Layouttransformer: Layout generation and completion with self-attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1004–1014, 2021. 1, 2

[16] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 7

[17] A. A. Jyothi, T. Durand, J. He, L. Sigal, and G. Mori. Layoutvae: Stochastic scene layout generation from a label set. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9895–9904, 2019. 1, 2

[18] M. Keshavarzi, A. Parikh, X. Zhai, M. Mao, L. Caldas, and A. Yang. Scenegen: Generative contextual scene augmentation using scene graph priors. *arXiv preprint arXiv:2009.12395*, 2020. 1, 3

[19] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 3

[20] J. Li, J. Yang, A. Hertzmann, J. Zhang, and T. Xu. Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767*, 2019. 1, 2

[21] M. Li, A. G. Patil, K. Xu, S. Chaudhuri, O. Khan, A. Shamir, C. Tu, B. Chen, D. Cohen-Or, and H. Zhang. Grains: Generative recursive autoencoders for indoor scenes. *ACM Transactions on Graphics (TOG)*, 38(2):1–16, 2019. 1, 3

[22] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021. 4

[23] A. Luo, Z. Zhang, J. Wu, and J. B. Tenenbaum. End-to-end optimization of scene layout. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3754–3763, 2020. 1, 3

[24] R. Ma, A. G. Patil, M. Fisher, M. Li, S. Pirk, B.-S. Hua, S.-K. Yeung, X. Tong, L. Guibas, and H. Zhang. Language-driven synthesis of 3d scenes from scene databases. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018. 3

[25] K. Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, 2002. 1, 2, 11

[26] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 3

[27] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022. 2, 4

[28] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021. 2, 4

[29] D. Ritchie, K. Wang, and Y.-a. Lin. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6182–6190, 2019. 1, 3

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 4

[31] K. Wang, Y.-A. Lin, B. Weissmann, M. Savva, A. X. Chang, and D. Ritchie. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)*, 38(4):1–15, 2019. 1, 3

[32] K. Wang, M. Savva, A. X. Chang, and D. Ritchie. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018. 3

[33] X. Wang, C. Yeshwanth, and M. Nießner. Sceneformer: Indoor scene generation with transformers. *arXiv preprint arXiv:2012.09793*, 2020. 4

[34] Z. Wu, X. Wang, D. Lin, D. Lischinski, D. Cohen-Or, and H. Huang. Sagnet: Structure-aware generative network for 3d-shape modeling. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019. 3

[35] L. Xu, Y. Xiangli, A. Rao, N. Zhao, B. Dai, Z. Liu, and D. Lin. Blockplanner: City block generation with vectorized graph representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5077–5086, 2021. 3

[36] C.-F. Yang, W.-C. Fan, F.-E. Yang, and Y.-C. F. Wang. Layouttransformer: Scene layout generation with conceptual and spatial diversity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3732–3741, 2021. 1, 2

[37] J. Yu, Y. Xu, J. Y. Koh, T. Luong, G. Baid, Z. Wang, V. Vasudevan, A. Ku, Y. Yang, B. K. Ayan, et al. Scaling autoregressive models for content-rich text-to-image generation. *arXiv preprint arXiv:2206.10789*, 2022. 2, 4

[38] L. F. Yu, S. K. Yeung, C. K. Tang, D. Terzopoulos, T. F. Chan, and S. J. Osher. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011, v. 30,(4), July 2011, article no. 86*, 30(4), 2011. 3

[39] H. Zhou, J. Sun, G. Turk, and J. M. Rehg. Terrain synthesis from digital elevation models. *IEEE transactions on visualization and computer graphics*, 13(4):834–848, 2007. 1

[40] Y. Zhou, Z. While, and E. Kalogerakis. Scenegraphnet: Neural message passing for 3d indoor scene augmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7384–7392, 2019. 3