Deep Learning Techniques for Digital Content Generation

by

Hang Chu

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Deep Learning Techniques for Digital Content Generation

Hang Chu
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2020

Digital content generation is in high demand, yet many important steps of it remain inefficient. We investigate various deep learning techniques that introduce automation into digital content generation. In the following chapters of this thesis, we introduce novel techniques for automatically generating various components of a virtual world: from houses and city layouts to animated conversations and realistic telepresence. We start with HouseCraft, where we exploit rental ads and a small set of StreetView images to build photorealistic 3D models of the buildings' exterior. It is followed by Neural Turtle Graphics, a novel generative model for spatial graphs, with various demonstrations of its applications in modeling city road layouts. We then introduce Face-to-Face Neural Conversation, a neural conversation model that aims to read and generate facial gestures alongside with text. Finally, we present Modular Codec Avatar, a method to generate expressive and hyper-realistic faces driven by the cameras in the VR headset.

In memory of my grandfather, Lianshou Zhang.

Acknowledgements

Acknowledgements to be added.

Contents

1 Introduction **1**

2 HouseCraft: Building Houses from Rental Ads and Street Views **2**

2.1 Abstract . . . 2
2.2 Introduction . . . 2
2.3 Related work . . . 3
2.3.1 Interactive Image-based Modeling . . . 3
2.3.2 Automatic 3D Building Modeling . . . 3
2.3.3 Facade Parsing . . . 4
2.3.4 Using Floorplans for Vision . . . 4
2.3.5 Holistic 3D Scene Understanding: . . . 5
2.4 The SydneyHouse Dataset . . . 5
2.4.1 Data Collection . . . 5
2.4.2 Ground-Truth Annotation . . . 6
2.4.3 Statistics . . . 6
2.5 Building Houses from Rentals Ads and Street Views . . . 6
2.5.1 Parameterization and Energy Formulation . . . 7
2.5.2 Potentials in our Energy . . . 8
2.5.3 Efficient Projection of Building Hypotheses . . . 10
2.5.4 Inference . . . 11
2.5.5 Learning . . . 11
2.6 Experiments . . . 11
2.6.1 Implementation details . . . 11
2.6.2 Evaluation . . . 12
2.7 Conclusion . . . 14

3 Neural Turtle Graphics for Modeling City Road Layouts **17**

3.1 Abstract . . . 17
3.2 Introduction . . . 17
3.3 Related Work . . . 18
3.3.1 Classical Work . . . 18
3.3.2 Generative Models of Graphs . . . 19
3.3.3 Graph-based Aerial Parsing . . . 19
3.3.4 Most Related Work . . . 19

3.4 Neural Turtle Graphics . . . 20
3.4.1 Notation . . . 20
3.4.2 Graph Generation . . . 20
3.4.3 Aerial Road Parsing . . . 21
3.4.4 Implementation Details . . . 22
3.4.5 Learning & Inference . . . 22
3.5 Experiments . . . 23
3.5.1 City Road Layout Generation . . . 23
3.5.2 Satellite Road Parsing . . . 27
3.5.3 Environment Simulation . . . 30
3.6 Conclusion . . . 30

4 A Face-to-Face Neural Conversation Model 31
4.1 Abstract . . . 31
4.2 Introduction . . . 31
4.3 Related Work . . . 33
4.3.1 Dialogue Systems . . . 33
4.3.2 Neural Conversation Models . . . 33
4.3.3 Face and Gesture Modeling . . . 34
4.4 The MovieChat Dataset . . . 34
4.5 Face-to-Face Neural Conversation Model . . . 35
4.5.1 FACS Gesture Representation . . . 36
4.5.2 Face-to-Face Conversation Model . . . 36
4.6 Experiments . . . 40
4.6.1 The Mind-Reading Test . . . 40
4.6.2 The NeuralHank Chatbot . . . 41
4.7 Conclusion . . . 43

5 Expressive Telepresence via Modular Codec Avatars 44
5.1 Abstract . . . 44
5.2 Introduction . . . 44
5.3 Related Work . . . 46
5.3.1 Morphable 3D Facial Models . . . 46
5.3.2 Deep Codec Avatars . . . 47
5.3.3 VR Telepresence . . . 47
5.4 Hardware and Dataset . . . 48
5.5 Method . . . 49
5.5.1 Model Formulation . . . 49
5.5.2 Exemplar-based Latent Alignment . . . 51
5.5.3 Modulated Adaptive Blending . . . 51
5.5.4 Implementation Details . . . 52
5.6 Experiments . . . 52
5.6.1 Facial Expression Modeling: Modular vs. Holistic . . . 52
5.6.2 Full VR Telepresence . . . 53

5.6.3 Extensive Applications 54
5.7 Conclusion 56

Bibliography **58**

List of Tables

2.1 Average values of vertical positions computed from GT annotations. 6
2.2 Mean errors of our approach and the baselines. 9
2.3 Object detection performance evaluation. 10
2.4 Impact of the size of the search range on the accuracy of the method. Columns from left to right: xy search range, percentage of ground truth that is within the search range, inference time per house, average error for each variable. 11
2.5 Average loss on the validation set for different hyper-parameters. 13
2.6 Ablation study of different types of energy terms. 13

3.1 Dataset statistics of RoadNet and SpaceNet [2]. 23
3.2 Perceptual domain-adapted FIDs ({maxpool1,maxpool2,pre-aux,fc}, lower is better), Urban Planning feature differences ({density,connectivity,reach,convenience}, lower is better), and Diversity evaluation of city generation. Ratings (higher is better) are computed by averaging with scales {10,10,10,20} for perceptual and {60,30,50,20} for urban planning. Extremely lowDiversity indicates incapability of creating new cities. 23
3.3 Comparison of methods on the standard SpaceNet split. 29
3.4 SpaceNet evaluation on unseen city by holding one city out in training. Without finetuning, the RoadNet pretrained NTG-P is able to improve over DLA+STEAL. 29

4.1 The mind-reading text results on text. Second column lists word *perplexity* (lower the better). Third to last columns list unigram *precision*, *recall*, and $F1$-*score* (higher the better) across different beam search size. For each column, we mark the **best** and **second best** results in red and blue color. We underscore the **overall best**result across all methods and all beam sizes. 37
4.2 The mind-reading test results on gesture. Legend same as Table 4.1. 38
4.3 AMT user study on interestingness and naturalness. The evaluation is conducted in form of pairwise comparison. We further accumulate number of votes for different methods. . . 43

5.1 Quantitative results of our main experiment evaluating the full VR telepresence system robustness. MCA outperforms CA across various metrics. Please refer to the text for details. 54
5.2 An ablation study of our method showing the progression and contribution to the overall performance improvement on the main RMSE metric. 55

List of Figures

2.1 We exploit rental ads and a small set of (wide-baseline) Google's StreetView images to build photo-realistic 3D models of the buildings' exterior. In particular, our method creates models by using only the (approximate) address and a floorplan extracted from the rental ad, in addition to StreetView. 3

2.2 House selection: (**left**) example of a house in the dataset, **(right)** shows a discarded ad since the house cannot be identified from StreetView images. 4

2.3 Getting the initial building position and StreetView images, and our annotation interface. In **(a)**, blue pin shows the geo-reference result, green pins show three nearby StreetView images $S_{1,2,3}$, red lines depict the building contours parsed from the map image. **(b)-(d)** show the StreetView images corresponding to $S_{1,2,3}$, respectively, overlaid with the virtual house rendered with the initial building position. 5

2.4 Visibility and parametrization: (a) shows a floorplan and the visible parts to a given camera, (b) depicts our parametrization of the building and its pose. 7

2.5 Feature visualization. (a) depicts the initial building position – all features are computed for a region surrounding the initial building as shown in cyan in (b). (b) detected vertical (blue), horizontal (green), and otherwise orientated (red) edges. (c) detected objects such as window (red) and door (green). (d) foreground segmentation with foreground and background color sampled from cyan and magenta regions. (e) saliency foreground segmentation. (f) semantic segmentation with *building* (red), *sky* (purple), possible *occlusion* (blue), and *other* (light yellow). 8

2.6 Failure modes. Left: original images. Right: our results. 10

2.7 Qualitative comparison. From left to right: input StreetView image, baseline method, our approach, texture-mapped 3D house model using our result. 15

2.8 Our approach demonstrates robustness when the house is occluded. Left: original images (three for each house). Right: our results. 16

3.1 We introduce Neural Turtle Graphics (NTG), a deep generative model for planar graphs. In the Figure, we show NTG's applications to (interactive) city road layout generation and parsing. 18

3.2 Illustration of the Neural Turtle Graphics (NTG) model. (a) depicts acyclic incoming paths $\{\mathbf{s}^{in}\}$ of an active node $\mathbf{v}_i$, each of which is encoded using an RNN encoder. NTG decoder then predicts a set of outgoing nodes $\{\mathbf{v}^{out}\}$. (b) shows the NTG's neural network architecture. First, the encoder GRU consumes the motion trajectory $\Delta\mathbf{x}^{in}$ of each incoming path. We produce an order-invariant representation by summing up the last-state hidden vectors across all paths. Next, the decoder produces "commands" to advance the turtle and produces new nodes. An optional attribute vector can be further added to the decoder depending on the task. 20

3.3 Qualitative examples of city road layout generation. GraphRNN-2D generates unnatural structures and fails to capture city style. PGGAN is unable to create new cities by either severely overfitting, or producing artifacts. CityEngine produces less style richness due to its fixed rule-based synthesis algorithm. NTG is able to both capture the city style and creating new cities. 24

3.4 City-wise FID (fc) of different methods. 25

3.5 NTG creates new city road layouts in a combinatorial manner. Local patterns as shown by orange boxes are remembered, and then intertwined to create novel structures. 26

3.6 Effect of sampled paths K and maximum path length L on reconstruction quality in meters (red), inference time in seconds per km^2 (green), and FID-fc (blue). 26

3.7 Examples of interactive city road layout generation via user sketching and local style selection. 27

3.8 NTG can be easily extended to generate road type. 28

3.9 Qualitative examples of SpaceNet road parsing. 28

3.10 Sat2Sim: converting satellite image into a series of simulated environments. Buildings and vegetation added via [1]. 29

4.1 Facial gestures convey sentiment information. Words have different meanings with different facial gestures. Saying "*Thank you*" with different gestures could either express gratitude, or irony. Therefore, a different response should be triggered. 32

4.2 Example conversations from our MovieChat dataset. Each row shows two examples, left shows query face and text, right shows target face and text. Our dataset has various conversation scenarios, such as simple conversations shown in the first and second rows on the left, as well as more challenging cases shown on the right. 32

4.3 Overview of our MovieChat database. (a) and (b) show an example frame with 3D face detection and detected FACS intensities. We obtain detections using the off-the-shelf OpenFace [10] package. (c) shows the scale of our MovieChat database. Our database is by far the largest language-face conversation video dataset. 33

4.4 List of gestures recorded in the MovieChat dataset, and percentage of frames where each gesture is dominant. 34

4.5 Facial Landmark (FL) systems. Left shows an example of "in the wild" landmarks [10, 28, 137], which fails to capture subtle gesture information. Right shows invasive motion capture landmarks [3]. 35

4.6 Our face-to-face conversation model. Our model consists of 6 RNNs shown in different colors. First, the text-face sequences of query and conversation history are encoded by text and face encoders (only one history sentence is depicted). History sentence encodings are further encoded by the history encoder. Next, encodings are added to form the context vector, which the text and face decoders are conditioned on. Finally, we generate frame-level, micro-gesture animation controls based on the word decodings. 35
4.7 Success and failure cases of using face along with text. Top five rows show successful examples where adding facial gesture information produces sentences closer to the ground truth. Bottom five rows show failure modes, including face detection failure in the sixth row, and detecting another face that does not belong to the speaker in the seventh row. . 40
4.8 NeuralHank examples. `Q` is the query text. `A1`, `A2`, and `A3` are generated by *noMicro-noGAN*, Micro-noGAN, and *Ours*, respectively. We also show one animation frame generated by our method. First two rows show that our GAN-based model generates more diverse and interesting responses. Last row shows failure cases where our method generates confusing responses. Please refer to our project page for animation videos, more examples, and chatting with Hank live through a webcam. 42

5.1 Train and test pipeline for our VR telepresence system. In the first stage, we capture facial expressions of a user using both multi-view camera dome and VR headset (mounted with face-looking cameras). Correspondences between VR headset recording and full face expressions are established using the method in [125]. Finally, once the person-specific face animation model is learned using these correspondences, a real-time photo-realistic avatar is driven from the VR headset cameras. 45
5.2 Model diagrams comparing the previous CA and the proposed MCA. K denote the number of head-mounted cameras. In CA, images of all headset cameras are feed together to the single encoder $\mathcal{E}$ to compute the full face code which is subsequently decoded into 3D face using deocoder $\mathcal{D}$. In MCA, image of each camera is encoded separately into modular code $\mathbf{c}_k^{\text{part}}$ with the encoder $\mathcal{E}_k$, which is feed to a synthesizer $\mathcal{S}_k$ to estimates a camera specific full face code $\mathbf{c}_k^{\text{full}}$ and blending weights. Finally all these camera specific full face codes are decoded into 3D faces and blended together to form the final full face. 46
5.3 Hardware and dataset examples. First row: capture dome [70], training headset, tracking headset, and sample images [125]. Second row: head-mounted camera examples of left eye, right eye, and mouth from different capture runs. Third row: examples of head-mounted images and ground-truth correspondences. Last row: dataset statistics. 48
5.4 Comparing holistic versus modular by isolating and evaluating the importance expressiveness. X-axis shows different capacities of the face expressions by the number of clusters. Y-axis shows the RMSE photometric error. Note that CA uses proportionally K times more clusters than each module of MCA for fair comparison. 53
5.5 Qualitative VR telepresence results. Compared to the holistic approach, MCA handles unseen subtle expressions better due to stronger expressiveness. 55
5.6 Qualitative results of MCA from different viewing directions by varying $\mathbf{v}$. MCA produces natural expressions that are consistent across viewpoints. 56
5.7 Failure cases of MCA. Typical failure cases include interference from strong background flash, extreme asymmetry between the eyes, and weakened motion. 56

5.8 Two new applications enabled by the MCA model. Left side shows two examples of flexible animation. Right side shows two examples of eye amplification. 56

Chapter 1

Introduction

The demand for digital content generation has been increasingly high in the past few decades. This is especially the case for the film, television, and video game production industry. In the film-making industry, visual effects are known to be expensive. For example, the movie Avengers 4 has a production budget of over 300 million US dollars, while the most recent remake of King Kong has a production budget of over 200 million US dollars. The recent video game Destiny has a record-breacking production budget of over 500 million US dollars, while it is common for major video games to have a production budget over 100 million US dollars such as Grand Theft Auto 5 and Red Dead Redemption 2.

In the production of blockbuster movies and video games, a significant proportion of production budget is spent on creating digital contents of various kinds, such as a realistic digital replication of an ancient city, a visually appealing creation of a multi-planet sci-fi environment, or accurate and convincing animations of character faces and bodies. However, the majority of current adigital content generation approaches are tedious and inefficient. For instance, the creation of a digital house by sketching over a photo and create a corresponding 3D model, the creation of virtual towns with procedural rules that requires complex tuning of algorithm parameters, and the creation of facial animation via combining blendshapes.

The motivation of this thesis is to introducing automation into the digital content generation process. Towards this direction, we aim to achieve three main goals. First is to assist content-creating artists by building automatic and/or interactive toolboxes, which provides efficient solutions to repetitive steps in their workflow. Second is to improve the quality of content generation, by offering a variety of diverse and realistic digital assets, which can be used as a rich source of inspiration. Third is to enable new forms of media. This is especially the case as the emergence of augmented reality and virtual reality technologies, which demands massive amount of digital content generation at unprecedented scale.

The organization of this thesis is as followed: The thesis is divided into the first half of city modeling, comprising chapters 2 for modeling houses, and chapter 3 for modeling city road layouts. The second half focuses on the modeling of human facial avatars, with chapter 4 for generating animated conversations, and chapter 5 for generating photo-realistic telepresence into the virtual world. The former and latter halves share the common objective of automatic generation of a digital virtual world in a layer-wise manner. In the first half we aim to create the static and environmental layers of cities and buildings, while in the second half we address the dynamic digital agents that resides in the virtual world.

Chapter 2

HouseCraft: Building Houses from Rental Ads and Street Views

2.1 Abstract

We exploit rental ads and a small set of (wide-baseline) Google's StreetView images to build photo-realistic 3D models of the buildings' exterior. In particular, our method creates models by using only the (approximate) address and a floorplan extracted from the rental ad, in addition to StreetView.

2.2 Introduction

Creating 3D models of building exteriors has become an important area of research with applications in 3D city modelling, virtual tours and urban planning. The problem entails estimating detailed 3D geometry of the building exterior as well as parsing semantically important facade elements such as windows and doors. Precise registration of the geometry with the street-level imagery is crucial in order to allow for photo-realistic texture mapping onto the 3D models.

Most current approaches to 3D building estimation typically require LIDAR scans from either aerial [117,134] or ground-level views [85], or ground-level video scans [129]. While these approaches have shown impressive results [85, 120, 129], their use is inherently limited to the availability of such sensors. In this paper, our goal is to enable a wider use, where the user can easily obtain a photo-realistic model of her/his house by providing only an approximate address and a floorplan of the building.

Towards this goal, we exploit rental ads which contain both the property's address as well as a floor plan. We convert the address to a rough geo-location, and exploit Google's Geo-Reference API to obtain a set of StreetView images where the property's exterior is visible. A floorplan provides us with an accurate and metric outline of the building's exterior along with information about the position of windows and doors. This information is given within the footprint of the building, and typically the vertical (along the height of the house) positions are not known.

Our approach then reasons jointly about the 3D geometry of the building and its registration with the Google's StreetView imagery. In particular, we estimate the height of each floor, the vertical positions of windows and doors, and the accurate position of the building in the world's map. We frame the problem as inference in a Markov random field that exploits several geometric and semantic cues from

Figure 2.1: We exploit rental ads and a small set of (wide-baseline) Google's StreetView images to build photo-realistic 3D models of the buildings' exterior. In particular, our method creates models by using only the (approximate) address and a floorplan extracted from the rental ad, in addition to StreetView.

the StreetView images as well as the floorplan. Note that the StreetView images have a very wide baseline, and thus relying on keypoint matching across the views would result in imprecise estimation. In our model, we exhaustively explore the solution space, by efficiently scoring our projected building model across all views. Our approach is thus also partially robust to cases where the building is occluded by vegetation.

To demonstrate the effectiveness of our approach, we collected a new dataset with 174 houses by crawling an Australian rental website. We annotated the precise pose of each house with respect to Google's StreetView images, as well as the locations of windows, doors and floors. Our experiments show that our approach is able to precisely estimate the geometry and location of the property. This enables us to reconstruct a photo-realistic 3D model of the building's exterior. We refer the reader to Fig. 2.1 for an illustration of our approach and an example of our 3D rendering. Our dataset and the source code will be made available upon acceptance.

2.3 Related work

2.3.1 Interactive Image-based Modeling

Interactive modeling of buildings use one or more images as a guide to build 3D models. In the seminal work of [30], users are required to draw edges over multi-view images and the architectural 3D model is then built by a grammar of parameterized primitive polyhedral shapes. Sinha et al. [102] proposed an interactive system for generating textured 3D architectural models by sketching multiple photographs. Camera poses are recovered by structure from motion and 3D models are constrained by vanishing point detection. We refer the reader to Musialski et al. [78] for a more systematic literature review.

2.3.2 Automatic 3D Building Modeling

Researchers also attempted to tackle the problem in a fully automatic way. Many approaches made use of LIDAR aerial imagery [117, 134] for this purpose. In our review, we focus on approaches that exploit ground-level information. The general idea is to utilize prior knowledge to constrain the target 3D models, such as parallelism, orthogonality, piece-wise planar and vanishing point constraints [32, 76, 101, 127, 129]. These methods either rely on dense point clouds reconstructed from multiview stereo [101, 129] or line

Figure 2.2: House selection: (**left**) example of a house in the dataset, **(right)** shows a discarded ad since the house cannot be identified from StreetView images.

segment [127] features that guide the architectural reconstruction. However, line segment based methods are not robust to clutter and occlusion, while multi-view approaches rely on the input to be either a video or a set of images with relative small motion. Unlike previous approaches, our method requires only on a very sparse set of large-baseline images (typically 3 images per house with average distance between the cameras of 16.67 meters). Furthermore, we can handle large occlusions and clutter, such as vegetation and cars in the street.

2.3.3 Facade Parsing

Our method is also related to work on facade parsing, which aims at semantic, pixel-level image labeling of the building's facade [26, 74, 77, 110]. Hidden structures of building facades are modeled and utilized to tackle this problem, such as repetitive windows, low-rankness and grammar constraints. Compared to these structures, our method exploits the geometry and semantic priors from the floorplan, and can thus work with a much wider range of facades.

2.3.4 Using Floorplans for Vision

Floorplans contain useful yet inexpensive geometric and semantic information. They have been exploited for 3D reconstruction [17, 39, 50] and camera localization [24, 69, 123]. However, past methods mainly utilize floorplans for indoor scenes. This is in contrast to our work which aims to exploit floorplans for outdoor image-based modeling. To the best of our knowledge, this has not been proposed in the literature before. Moreover, due to the rich information available in the floorplans, our 3D parameterized model is very compact with a relative small number of degrees of freedom.

Figure 2.3: Getting the initial building position and StreetView images, and our annotation interface. In **(a)**, blue pin shows the geo-reference result, green pins show three nearby StreetView images $S_{1,2,3}$, red lines depict the building contours parsed from the map image. **(b)-(d)** show the StreetView images corresponding to $S_{1,2,3}$, respectively, overlaid with the virtual house rendered with the initial building position.

2.3.5 Holistic 3D Scene Understanding:

It has been shown that semantics and geometry can benefit each other. Holistic models aim at solving many of these tasks simultaneously. For instance, [57, 92] perform semantic parsing and multi-view reconstruction jointly and [58,68,122] reason about both depth and semantic labeling from a monocular image. Our approach also reasons about semantic parsing and pose estimation jointly, but constrains the degrees of freedom via floorplan priors.

2.4 The SydneyHouse Dataset

We exploit rental ads to obtain our dataset. In this section we first explain data collection and analyze the statistics of our new dataset.

2.4.1 Data Collection

We collected our dataset by crawling an Australian real-estate website[1]. We chose this site because it contains housing information in a formatted layout. We queried the website by using the keyword "Sydney"+"House", and parsed ads for the top 1,007 search results. A rental ad is typically composed of several, mostly indoor photos, a floorplan, as well as meta data information such as the address of the property. In our work we only kept houses for which a floorplan was available (80.8%). Given the address, we obtained several outdoor street-level photos around the property via Google's Geo-Reference API[2]. In particular, for each rental ad we downloaded a local map around the property using Google Maps, as depicted in Fig. 2.3(a). We then parse the building contours from the map using standard

[1] http://www.domain.com.au/
[2] https://www.google.com/maps

Vertical position	base	floor	door	upper window	lower window
Mean value	-2.57m	3.33m	2.56m	0.53m	1.34m

Table 2.1: Average values of vertical positions computed from GT annotations.

image processing techniques[3], and set the building position as the position of the closest building contour. This gives us a very rough geo-location of the rental property. We then collected Google StreetView images around this location, where we set the camera orientation to always point towards the property of interest. Fig. 2.3(b)-(d) show that although the position of the building matches the map very well, the property does not align well with the image. Obtaining a more accurate alignment is the subject of our work. We finally discarded ads for which the houses could not be geo-referenced or were not identifiable by the annotators in the StreetView imagery (65.3%). Fig. 2.2 shows examples of house selection. Our final *SydneyHouse* dataset contains 174 houses located in different parts of Sydney.

2.4.2 Ground-Truth Annotation

We annotated the data with the vertical positions of the house (e.g., floor height, door height) and its precise location with respect to the geo-tagged imagery. We developed a WebGL based annotation tool that constructs the 3D model of the house given a set of vertical positions. The tool visualizes the projection of the 3D model onto the corresponding StreetView images. The in-house annotators were then asked to adjust the properties until the 3D model is best aligned with the streetview imagery. For each house, we annotate the 1) vertical positions, namely the building's foundation height (w.r.t. camera), 2) each floor's height as well as 3) the door heights, and 4) windows vertical starting and ending positions. We also annotate the floorplan by specifying line segments for walls, windows, and doors, as well as building orientation and scale. On average, the total annotating time for each house was around 20 minutes.

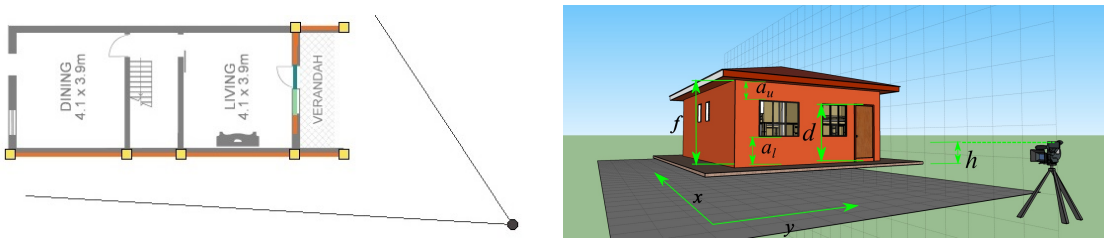
2.4.3 Statistics

Our dataset consists on 174 houses. On average each house has 1.43 levels, 3.15 windows and 1.71 doors. Average values of vertical positions are listed in Table 2.1.

2.5 Building Houses from Rentals Ads and Street Views

In this section, we show how to create a 3D model of the building exterior given a floor plan and a set of wide-baseline StreetView images. We start by describing our parametrization of the problem in terms of the random variables denoting the building's position and vertical positions. We cast the problem as energy minimization in a Markov Random Field, where inference can be performed efficiently despite a large combinatorial solution space.

[3]In particular, we use a processing pipeline consisting of color thresholding, connected component analysis, corner detection, and sorting by geodesic distance.



(a) (b)

Figure 2.4: Visibility and parametrization: (a) shows a floorplan and the visible parts to a given camera, (b) depicts our parametrization of the building and its pose.

2.5.1 Parameterization and Energy Formulation

Given several geo-tagged StreetView images of a house, $\mathcal{I} = \{\mathcal{I}_i\}_{i=1}^{N}$, and a floorplan $\mathcal{F}$ extracted from the rental ad, our goal is to jointly estimate the 3D layout of the house as well as its accurate geo-location. A floorplan contains information about the number of floors, the dimensions of the footprint of each floor, and the (2D) location of doors and windows in the footprints. In order to lift the floorplan to 3D, we need to estimate the height of each floor, the building's foundation height as well as the vertical position of each door, window, and possibly a garage gate. Let h be the building's foundation height, where h=0 means that the building sits on a plane having the same height as the camera. Here, h thus simply encodes the vertical offset of the building's support surface from the camera. We parameterize all floors with the same height, which we denote by f. We also assume that all doors (including the garage gates) have the same height, which we denote by d. Further, let $\mathbf{a} = \{a_u, a_l\}$ be the window's vertical starting and ending position.

Our initial estimate of the house's location is not very accurate. We thus parameterize the property's true geolocation with two additional degrees of freedom (x, y), encoding the (2D) position of the house in the map. Note that this parameterization is sufficient as projection errors are mainly due to poor geo-location in google maps and not because of inaccurate camera estimates in Google StreetView. We confirmed this fact when labeling our new dataset. Fig. 2.4 visualizes our full parametrization.

Let $\mathbf{y} = \{x, y, h, f, d, \mathbf{a}\}$ be the set of all the variables we are interested in estimating for a single house. We formulate the problem as inference in a Markov random field, which encodes the fact that the projection of the model should match the edges, semantics and location of doors and windows in all images. Furthermore, we would like to encourage the building to be salient in the image, and its appearance to be different than the one of the background. Our complete energy takes the following form:

$$\begin{aligned} E(\mathbf{y};\mathcal{I},\mathcal{F}) = E_{\text{edge}}(\mathbf{y};\mathcal{I},\mathcal{F}) + E_{\text{obj}}(\mathbf{y};\mathcal{I},\mathcal{F}) + E_{\text{seg}}(\mathbf{y};\mathcal{I},\mathcal{F}) \\ + E_{\text{sal}}(\mathbf{y};\mathcal{I},\mathcal{F}) + E_{\text{app}}(\mathbf{y};\mathcal{I},\mathcal{F}) \end{aligned} \tag{2.1}$$

In the following subsection, we describe the potentials in more detail.

We first note that given a building hypothesis $\mathbf{y}$, our energy scores its projection in the set of StreetView images. Thus, in order to properly score a hypothesis, we need to reason about the visibility of the walls, windows, etc. We compute the visibility with a standard exact 2D visibility reasoning method [111]. Fig. 2.4(a) shows an example of visibility reasoning. We refer the reader to supplemental material for details.

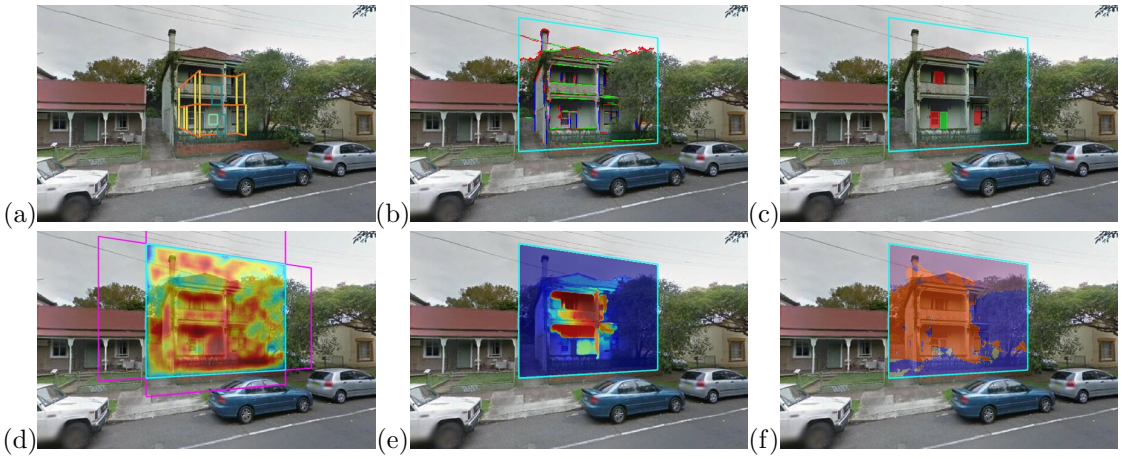


Figure 2.5: Feature visualization. (a) depicts the initial building position – all features are computed for a region surrounding the initial building as shown in cyan in (b). (b) detected vertical (blue), horizontal (green), and otherwise orientated (red) edges. (c) detected objects such as window (red) and door (green). (d) foreground segmentation with foreground and background color sampled from cyan and magenta regions. (e) saliency foreground segmentation. (f) semantic segmentation with *building* (red), *sky* (purple), possible *occlusion* (blue), and *other* (light yellow).

2.5.2 Potentials in our Energy

Edge Potential

This term encodes the fact that most building facade, window, door and floor boundaries correspond to image edges. We thus define

$$E_{\text{edge}}(\mathbf{y};\mathcal{I},\mathcal{F}) = \sum_{i=1}^{N} \mathbf{w}_{\text{edge}}^{T} \phi_{\text{edge}}(\mathbf{y};\mathcal{I}_i,\mathcal{F})$$

where N is the number of StreetView images, $\mathbf{w}_{\text{edge}}$ is a vector of learned weights and ϕ_{edge} is a feature vector encoding the distances between the edges of our projected hypothesis and the image edges. We compute distances for different types of building edges, i.e., *vertical*, *horizontal*, and *all*. When computing the potential, we take into account visibility and exploit distance transforms for efficiency. We use four different features per edge type, corresponding to four different thresholded distance transforms (i.e., 0.1m, 0.2m, 0.4m, all). The potential then sums for each edge type the value of the different distance transforms along the projected visible edges. In particular, we use structured edge [33] to extract the orientated edge map for each image.

Object Potential

To compute the object potential, we first run an object detector for three classes, i.e. *doors*, *windows*, and *garage gates*, on the StreetView images. This energy term is then designed to encourage agreement between the projection of the model's visible doors, windows, and garage gates, and the detection boxes. We thus define

$$E_{\text{obj}}(\mathbf{y};\mathcal{I},\mathcal{F}) = \sum_{i=1}^{N} \mathbf{w}_{\text{obj}}^{T} \phi_{\text{obj}}(\mathbf{y};\mathcal{I}_i,\mathcal{F})$$

	xy/m	IOU	h/cm	f/cm	d/cm	a_l/cm	a_u/cm
random	9.07	21.04%	102.8	49.8	45.6	47.0	55.9
box-reg [88, 107]	6.68	33.31%					
google	5.01	43.46%					
ours	**2.62**	**68.29%**	**49.7**	**43.1**	**14.1**	**36.9**	**33.6**

Table 2.2: Mean errors of our approach and the baselines.

where N is the number of images, $\mathbf{w}_{\text{obj}}$ is a vector of weights and ϕ_{obj} is a feature vector that encodes agreement for each object type. In particular, ϕ_{obj} simply counts the number of pixels for which the projected (visible) object hypothesis and the image detection box agree. We additionally use counts of pixels that are inside the projected object hypothesis, but are not in any detection box, and another count for pixels contained in the detection boxes but not contained inside the object hypothesis. Our feature vector is thus 9-dimensional (three classes and three counting features for each class). We refer the reader to Fig. 2.5(c) for an illustration. Note that these pixel counts can be computed efficiently with integral geometry [95], a generalization of integral images to non axis-aligned plane homographies. In our work, we rectify the image using the known homography to frontal-parallel view, so that the grid for integral geometry is axis-aligned, thus simplifying the implementation.

To detect doors, windows and garage gates in the StreetView images we use a pipeline similar to RCNN [42]. We first use edgebox [139] to generate object proposals due to its ability to handle rectangular objects. In particular, we use only 10 object proposals. We train a convolutional neural network on the region proposals for each class independently, by fine-tuning AlexNet [56] pre-trained on ImageNet [91], which is available in the Caffe's [51] model-zoo.

Segmentation Potential

This term encodes the fact that we prefer house configurations that agree with semantic labels extracted from StreetView images. Towards this goal, we take advantage of SegNet [8] and for each StreetView image compute semantic segmentation in terms of four classes: *sky*, *building*, *occlusion* and *other*. We define all categories that can occlude a building as *occlusion*, i.e., tree, pole, sign and vehicle. We then define our segmentation potential as

$$E_{\text{seg}}(\mathbf{y}; \mathcal{I}, \mathcal{F}) = \sum_i \mathbf{w}_{\text{seg}}^T \phi_{\text{seg}}(\mathbf{y}; \mathcal{I}_i, \mathcal{F})$$

where $\mathbf{w}_{\text{seg}}$ is a vector of weights and ϕ_{seg} is a feature vector. Each dimension of ϕ_{seg} counts the number of pixels inside the projected building region, that were labeled with one of the categories by the segmentation algorithm. We expect the learning algorithm to learn that the weight for building is positive and the weight for sky and other is negative. Similar to the object potential, this term can be efficiently computed via 2D integral geometry in a rectified frontal-parallel view.

Saliency Potential

This term encourages the building facade to correspond to salient objects in the scene. In particular, we use [135] to compute a per-pixel saliency score. We then compute our potential by simply summing the salient scores inside the projected building's region. Note that as before we can compute this potential

	Proposal		Detection			
	precision	recall	precision	recall	F1 score	overall acc.
window	10.11%	63.04%	34.65%	76.11%	47.62	84.10%
door	2.32%	32.50%	4.41%	39.29%	7.93	76.92%
garage	7.28%	49.43%	32.14%	81.82%	46.15	90.87%

Table 2.3: Object detection performance evaluation.

Figure 2.6: Failure modes. Left: original images. Right: our results.

in constant time using integral accumulators.

Appearance Potential

This term encourages the image region corresponding to the projected building's facade to have color different than the background. We sample the foreground and background color from image region around, and sufficiently far away, from the initial building position, where background normally corresponds to sky, pavement, grass, and other buildings. We compute per-pixel foreground probability with a kernel density estimator. Our potential is obtained by summing the foreground probability inside the projected facade.

2.5.3 Efficient Projection of Building Hypotheses

As shown in Fig. **??**(b), given a configuration $\mathbf{y}$, we can lift the floorplan $\mathcal{F}$ to 3D, and project it onto each StreetView image $\mathcal{I}_i$ given the known camera parameters (given by Google's API). Since the camera poses are fixed (we are "moving" the building relative to the camera), we can re-write the model's translation to point $\mathbf{z}$ as: $HK[R|t]\mathbf{z} = HK[R|t](\mathbf{z}_0 + x\triangle\mathbf{z}_x + y\triangle\mathbf{z}_y)$, where H is a homography that rectifies the image to the frontal-parallel view. Here $\mathbf{z}_0$ is a chosen initial point, $(\mathbf{z}_x, \mathbf{z}_y)$ is a chosen discretization of the search space, and (x, y) represents the coordinate of our hypothesis in this space.

range/m	gt%	time/s	xy/m	h/cm	f/cm	a_l/cm	a_u/cm
5×5	19.0%	**6.22**	3.37	58.1	41.7	41.2	**28.4**
10×10	62.1%	8.58	2.70	**44.0**	43.3	41.5	29.6
15×15	88.5%	12.12	2.66	49.2	42.0	37.1	33.3
20×20	96.6%	19.05	**2.62**	49.7	43.1	36.9	33.6
25×25	**97.7%**	27.16	2.67	46.6	**40.4**	**35.1**	34.2

Table 2.4: Impact of the size of the search range on the accuracy of the method. Columns from left to right: xy search range, percentage of ground truth that is within the search range, inference time per house, average error for each variable.

Since $\mathbf{z}_0$ and $(\mathbf{z}_x, \mathbf{z}_y)$ are fixed for all hypotheses, we can pre-compute $HK[R|t]\mathbf{z}_0$, $HK[R|t]\mathbf{z}_x$, and $HK[R|t]\mathbf{z}_y$, allowing us to avoid matrix multiplication when projecting a new hypothesis $\mathbf{z}$.

2.5.4 Inference

We perform inference by minimizing the energy in Eq. (2.1) with respect to $\mathbf{y}$. It is worth noting that our energy evaluation can be done very efficiently, since all the energy potentials can be computed via integral accumulators. On average we can evaluate 20.7k candidates per second on a CPU. For random variables corresponding to vertical building dimensions, we discretize our solution space by learning a set of prototypes by independently clustering the output space for each random variable. We then perform exhaustive search over the combinatorial space of all prototypes. We refer the reader to the supplementary material for a detailed description of our discretization.

2.5.5 Learning

We learn the parameters of the model using structured-SVMs [115]. In particular, we use the parallel implementation of [94]. We compute the loss function as the sum of loss functions across all StreetView images. For each individual image, we compute the loss as the intersection-over-union (IOU) between the ground-truth segmentation (implied by the ground-truth layout of the building projected into the image) and the segmentation corresponding to the model's projection. We weigh each class (i.e., building, window, doors/gates) differently. We estimate these weights as well as the importance of the regularization via cross-validation.

2.6 Experiments

We first provide implementation details, and then provide evaluation of our approach on our newly collected dataset.

2.6.1 Implementation details

For the building's position, we use a search range of 20m×20m, discretized by 0.25m. We use k-means clustering with a cluster variance of 0.2m for our variable-wise clustering, yielding 5 prototypes for h, 3 for f, 1 for d, 2 for a_l, and 2 for a_u. Thus in total, the number of unique solutions is $81 \times 81 \times 60$, which is roughly 0.4 million possible states.

2.6.2 Evaluation

We evaluate our approach on our SydneyHouse dataset. We conduct 6-fold evaluation, where for each test fold, we use 4 folds for training and 1 fold for validation to choose the hyper-parameters. We use grid search to choose the hyper-parameters over the space $c \in \{2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}\}$, and $\alpha' = \{0.5, 1, 2\}$, which is the ratio of object IOU to facade IOU in the task loss.

We compare our approach with three baselines. In the first baseline, we randomly generate the building position, with the same xy search range and discretization. We denote this baseline as *random*. In the second baseline, we feed the frontal-parallel facade image to Inception Network [107] and extract features from both global pooling and fully connected layers. We then follow the box regression process in [88] to find the optimal building bounding box, and choose the regularization parameter that yields best results. We then use the new box to obtain the building position xy. This baseline is referred to as *box-reg*. In the third baseline referred to as *google*, we obtain building position xy by matching Google street map building contour. In all baselines, vertical positions h, f, d, a_u, and a_l are obtained by randomly selecting from the training set. Baselines are repeated 1000 times to get the average performance.

Quantitative Results

As shown in Tab. 2.2 the box-regression baseline achieves better building position than the random baseline. However, its performance is still poor, due to limited number of training samples. Our method outperforms all baselines by a large margin. To better demonstrate our method's advantage, we also list the frontal facade IOU with the ground truth in Tab. 2.2. Note that our approach significantly improves the overlap with the ground truth.

Object Detection

As shown in Tab. 2.3 the object proposals detect a fraction of all objects. This is due to the fact that in our multi-view setup, only a few views face the frontal facade perpendicularly, and the rectified images are skewed and blurred in the other views. Doors are more difficult to detect than windows and garage gates as many doors are covered by the porch.

Impact of Search Space Discretization

We study the impact of the size of the search range for the building's xy position in Tab. 2.4. There is a trade-off between efficiency and efficacy. As we increase the search range, the percentage of ground-truth samples located within the search range increases, and building position accuracy also increases. However, above a certain range the accuracy starts to drop since more ambiguous samples are included.

Sensitivity to Hyper-Parameters and Ablation Study

We show detailed results for different hyper-parameters in Tab. 2.5 and an ablation study in Tab. 2.6. Note that incorporating more potentials increases the accuracy of our approach. Specifically, we can see that *other* potential, which includes segmentation, saliency, and appearance, helps to estimate the building position xy the most. The *edge* potentials are more useful for estimating the building foundation height h. The best result is achieved when combining all potentials.

α' \ $log_2 c$	-4	-3	-2	-1	0	1
0.5	0.515	0.539	0.517	0.502	0.502	0.508
1	0.571	0.585	0.558	0.562	0.573	0.561
2	0.651	0.637	0.619	0.617	0.626	0.613

Table 2.5: Average loss on the validation set for different hyper-parameters.

edge	obj	other	xy/m	h/cm	f/cm	a_l/cm	a_u/cm
✓			6.42	63.4	47.1	46.7	40.3
	✓		7.24	83.3	52.5	38.0	45.4
		✓	3.50	79.5	41.2	46.7	40.3
✓	✓		5.93	59.7	44.7	37.0	40.9
✓		✓	2.84	53.9	**40.8**	46.7	40.3
	✓	✓	3.18	60.6	46.4	37.1	**33.2**
✓	✓	✓	**2.62**	**49.7**	43.1	**36.9**	33.6

Table 2.6: Ablation study of different types of energy terms.

Qualitative Results

Fig. 2.7 shows a few qualitative results of our approach and the *google* baseline algorithm. It can be seen that the baseline cannot localize the house precisely in the image due to the mapping and geo-referencing errors. In contrast, our approach is able to provide accurate xy as well as vertical positions (floor/base heights, vertical window and door positions). Fig. 2.8 further shows four examples with partial occlusions caused by trees in one or two viewpoints. Despite occlusion, our approach is still able to accurately estimate the building position and vertical positions.

Failure Modes

Fig. 2.6 shows a few failure modes. In property 1, our approach fails because the initial building position is too noisy. The ground truth is 16.3m from the initial position, which exceeds our 20×20 search range. Property 2 shows another difficult case, where the building is heavily occluded in the second and third view, the facade has similar color to the sky, and many non-building edges exist. In this case, our method still estimates the building xy position correctly, but fails to estimate the vertical positions.

Inference Time

We report the efficiency of our method. Computing detection/segmentation features is done on a GPU, while the rest of code is all executed in CPU. Our CPU implementation uses Matlab without parallelization. Our approach takes 19.05s in total per house, which includes 3.41s for computing the image features, 8.00s for rendering all configurations and 7.64s for inference. Note that in our case both rendering and inference are highly paralelizable, thus allowing for high speed-ups with a more sophisticated implementation.

2.7 Conclusion

In this paper, we proposed an approach which exploits rentals ads to create photo-realistic 3D models of building exteriors. In particular, the property's address is employed to obtain a set of wide-baseline views of the building, while the floor plan is exploited to provide a footprint of the building's facade as well as the location on the floor of doors and windows. We formulated the problem as inference in a Markov random field that exploits several geometric and semantic cues from the StreetView images as well as the floorplan. Our experiments showed that our approach is able to precisely estimate the geometry and location of the property, and can create realistic 3D models of the building exterior.

(Original) (Google) (Ours) (3D model)

Figure 2.7: Qualitative comparison. From left to right: input StreetView image, baseline method, our approach, texture-mapped 3D house model using our result.

Figure 2.8: Our approach demonstrates robustness when the house is occluded. Left: original images (three for each house). Right: our results.

Chapter 3

Neural Turtle Graphics for Modeling City Road Layouts

3.1 Abstract

We propose Neural Turtle Graphics (NTG), a novel generative model for spatial graphs, and demonstrate its applications in modeling city road layouts. Specifically, we represent the road layout using a graph where nodes in the graph represent control points and edges in the graph represent road segments. NTG is a sequential generative model parameterized by a neural network. It iteratively generates a new node and an edge connecting to an existing node conditioned on the current graph. We train NTG on Open Street Map data and show that it outperforms existing approaches using a set of diverse performance metrics. Moreover, our method allows users to control styles of generated road layouts mimicking existing cities as well as to sketch parts of the city road layout to be synthesized. In addition to synthesis, the proposed NTG finds uses in an analytical task of aerial road parsing. Experimental results show that it achieves state-of-the-art performance on the SpaceNet dataset.

3.2 Introduction

City road layout modeling is an important problem with applications in various fields. In urban planning, extensive simulation of city layouts are required for ensuring that the final construction leads to effective traffic flow and connectivity. Further demand comes from the gaming industry where on-the-fly generation of new environments enhances user interest and engagement. Road layout generation also plays an important role for self-driving cars, where diverse virtual city blocks are created for testing autonomous agents.

Although the data-driven end-to-end learning paradigm has revolutionized various computer vision fields, the leading approaches [83] (e.g., the foundation piece in the commercially available CityEngine software) for city layout generation are still largely based on procedural modeling with hand-designed features. While these methods guarantee valid road topologies with user specified attribute inputs, the attributes are all hand-engineered and inflexible to use. For example, if one wishes to generate a synthetic city that resembles e.g. London, tedious manual tuning of the attributes is required in order to get plausible results. Moreover, these methods cannot trivially be used in aerial road parsing.

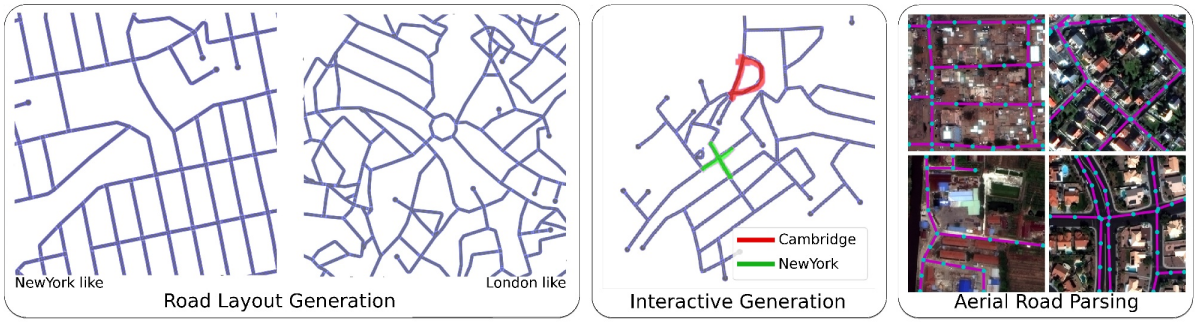


Figure 3.1: We introduce Neural Turtle Graphics (NTG), a deep generative model for planar graphs. In the Figure, we show NTG's applications to (interactive) city road layout generation and parsing.

In this paper, we propose a novel generative model for city road layouts that learns from available map data. Our model, referred to as *Neural Turtle Graphics* (NTG) is inspired by the classical turtle graphics methodology [1] that progressively grows road graphs based on local statistics. We model the city road layout using a graph. A node in the graph represents a spatial control point of the road layout, while the edge represents a road segment. The proposed NTG is realized as an encoder-decoder architecture where the encoder is an RNN that encodes local incoming paths into a node and the decoder is another RNN that generates outgoing nodes and edges connecting an existing node to the newly generated nodes. Generation is done iteratively, by pushing newly predicted nodes onto a queue, and finished once all nodes are visited.

Our NTG can generate road layouts by additionally conditioning on a set of attributes, thus giving control to the user in generating the content. It can also take a user specified partial sketch of the roads as input for generating a complete city road layout. Experiments with a comparison to strong baselines show that our method achieves better road layout generation performance in a diverse set of performance metrics. We further show that the proposed NTG can be used as an effective prior for aerial map parsing, particularly in cases when the imagery varies in appearance from that used in training. Fine-tuning the model jointly with CNN image feature extraction further improves results, outperforming all existing work on the Spacenet benchmark.

3.3 Related Work

3.3.1 Classical Work

A large body of literature exists on procedural modeling of streets. The seminal early work of [83] proposed an L-system which iteratively generates the map while adjusting parameters to conform to user guidance. This method became the foundation of the commercially available state-of-the-art CityEngine [1] software. Several approaches followed this line of work, exploiting user-created tensor fields [22], domain splitting [130], constraints stemming from the terrain [12, 40], and blending of retrieved examplars [7, 37, 81]. Methods that evolve a road network using constraints driven by crowd behaviour simulation have also been extensively studied [38, 84, 116].

[1] Turtle graphics is a technique for vector drawing, where a relative cursor (turtle) receives motion commands and leave traces on the canvas.

3.3.2 Generative Models of Graphs

Graph generation with neural networks has only recently gained attention [16, 65, 100, 131]. [131] uses an RNN to generate a graph as a sequence of nodes sorted by breadth-first order, and predict edges to previous nodes as the new node is added. [100] uses a variational autoencoder to predict the adjacency and attribute matrices of small graphs. [65] trains recurrent neural network that passes messages between nodes of a graph, and generates new nodes and edges using the propagated node representation. Most of these approaches only predict graph topology, while in our work we address generation of spatial graphs. Producing valid geometry and topology makes our problem particularly challenging. Our encoder shares similarities with node2vec [43] which learns node embeddings by encoding local connectivities using random walks. Our work focuses on spatial graph generation and particularly on road layouts, thus different in scope and application.

3.3.3 Graph-based Aerial Parsing

Several work formulated road parsing as a graph prediction problem. The typical approach relies on CNN road segmentation followed by thinning [75]. To deal with errors in parsing, [75] proposes to reason about plausible topologies on an augmented graph as a shortest path problem. In [66], the authors treat local city patches as a simply connected maze which allows them to define the road as a closed polygon. Road detection then follows Polygon-RNN [5,21] which uses an RNN to predict vertices of a polygon. [48] performs lane detection by predicting polylines in a top-down LIDAR view using a hierarchical RNN. Here, one RNN decides on adding new lanes, while the second RNN predicts the vertices along the lane. In our work, we predict the graph directly. Since our approach is local, it is able to grow large graphs which is typically harder to handle with a single RNN. Related to our work, [5, 23, 67, 73] annotate building footprints with a graph generating neural network. However, these works are only able to handle single cycle polygons.

3.3.4 Most Related Work

Most related to our work is RoadTracer [11], which iteratively grows a graph based on image evidence and local geometry of the already predicted graph. At each step, RoadTracer predicts a neighboring node to the current active node. Local graph topology is encoded using a CNN that takes as input a rendering of the existing graph to avoid falling back. Our method differs in the encoder which in our case operates directly on the graph, and the decoder which outputs several outgoing nodes using an RNN which may better capture more complex road intersection topologies. Furthermore, while [11] relied on carefully designed dynamic label creation during training to mimic their test time graph prediction, our training regime is simple and robust to test time inference.

We also note that with some effort many of these work could be turned into generative models, however, ours is the first that showcases generative and interactive modeling of roads. Importantly, we show that NTG trained only on map data serves as an efficient prior for aerial road parsing. This cannot easily be done with existing work [11, 66] which all train a joint image and geometry representation.

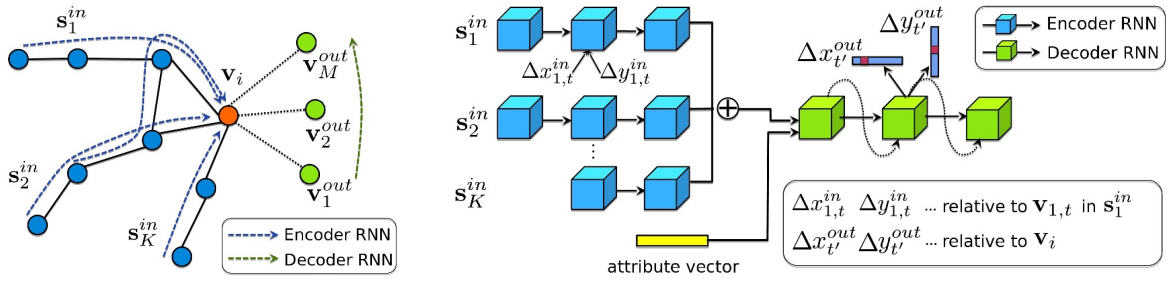


Figure 3.2: Illustration of the Neural Turtle Graphics (NTG) model. (a) depicts acyclic incoming paths $\{\mathbf{s}^{in}\}$ of an active node $\mathbf{v}_i$, each of which is encoded using an RNN encoder. NTG decoder then predicts a set of outgoing nodes $\{\mathbf{v}^{out}\}$. (b) shows the NTG's neural network architecture. First, the encoder GRU consumes the motion trajectory $\Delta\mathbf{x}^{in}$ of each incoming path. We produce an order-invariant representation by summing up the last-state hidden vectors across all paths. Next, the decoder produces "commands" to advance the turtle and produces new nodes. An optional attribute vector can be further added to the decoder depending on the task.

3.4 Neural Turtle Graphics

We formulate the city road layout generation problem as a planar graph generation problem. We first introduce the notation in Sec. 3.4.1 and then describe our NTG model in Sec. 3.4.2. Aerial parsing, implementation details, training and inference are given in Sec. 3.4.3-Sec. 3.4.5, respectively.

3.4.1 Notation

Road Layout

We represent a city road layout using an undirected graph $G = \{V, E\}$, with nodes V and edges E. A node $\mathbf{v}_i \in V$ encodes its spatial location $[x_i, y_i]^T$, while an edge $e_{\mathbf{v}_i,\mathbf{v}_j} \in \{0, 1\}$ denotes whether a road segment connecting nodes $\mathbf{v}_i$ and $\mathbf{v}_j$ exists. City road graphs are planar since all intersections are present in V. We assume it is connected, i.e. there is a path in G between any two nodes in V. The coordinates x_i and y_i are measured in meters, relative to the city's world location.

Incoming Paths. For a node $\mathbf{v}_i$, we define an *Acyclic Incoming Path* as an ordered sequence of unique, connected nodes which terminates at $\mathbf{v}_i$: $\mathbf{s}^{in}$= $\{\mathbf{v}_{i,1}, \mathbf{v}_{i,2}, ..., \mathbf{v}_{i,L}, \mathbf{v}_i\}$ where $e_{\mathbf{v}_{i,t},\mathbf{v}_{i,t+1}} = 1$ for each $1 \leq t < L$, and $e_{\mathbf{v}_{i,L},\mathbf{v}_i} = 1$, with L representing the length of the path. Since multiple different acyclic paths can terminate at $\mathbf{v}_i$, the set of these paths is denoted as $S_i^{in} := \{\mathbf{s}_k^{in}\}$.

Outgoing Nodes

We define $V_i^{out} := \{\mathbf{v}_j : \mathbf{v}_j \in V \wedge e_{\mathbf{v}_i,\mathbf{v}_j} = 1\}$, i.e. as the set of nodes with an edge to $\mathbf{v}_i$.

3.4.2 Graph Generation

We learn to generate graphs in an iterative manner. The graph is initialized with a root node and a few nodes connected to it, which are used to initialize a queue Q of unvisited nodes. In every iteration, an unvisited node from Q is picked to be expanded (called *active node*). Based on its *current local topology*, an encoder model generates a latent representation, which is used to generate a set of neighboring nodes

using a decoder model. These generated nodes are pushed to Q. The node to be expanded next is picked by popping from Q, until it is empty.

By construction, an active node $\mathbf{v}_i$ has at least one neighbor node in the graph. NTG extracts a representation of its local topology by encoding *incoming paths* S_i^{in} (of maximum length L) and uses the representation to generate a set of *outgoing nodes* V_i^{out} (if any) with edges to $\mathbf{v}_i$. These paths are encoded in an order-invariant manner and the resulting latent representation is used to generate a set of outgoing nodes V_i^{out}. NTG performs the encoding and decoding to generate the graph as described above with an encoder-decoder neural network. Fig. **??**(a) visualizes the process, while Fig. **??**(b) illustrates the encoder-decoder neural architecture, which is described in detail in the following sections.

NTG Encoder

We encode a single incoming path $\mathbf{s}^{in}$ into node $\mathbf{v}_i$ with a zero-initialized, bidirectional GRU [25]. The input to the GRU while processing the t^{th} node in the path is the motion vector between the nodes $\mathbf{v}_{i,t}^{in} \in \mathbf{s}^{in}$ and $\mathbf{v}_{i,t+1}^{in} \in \mathbf{s}^{in}$ in the path; i.e., $[\Delta x_{i,t}^{in}, \Delta y_{i,t}^{in}]^T = [x_{i,t+1}^{in}, y_{i,t+1}^{in}]^T - [x_{i,t}^{in}, y_{i,t}^{in}]^T$. This offset could be encoded as a discrete or continuous value, as discussed in Sec. **??**. The final latent representation $\mathbf{h}_{\text{enc}}$ for all paths is computed by summing the last hidden states of each path. Optionally, we append an attribute vector $\mathbf{h}_{\text{attr}}$ to the latent representation. For example, the attribute could be an embedding of a one-hot vector, encoding the city identity. This enables NTG to learn an embedding of city, enabling conditional generation. The final encoding is their concatenation $[\mathbf{h}_{\text{enc}}, \mathbf{h}_{\text{attr}}]$.

Sampling Incoming Paths

During training, for an active node $\mathbf{v}_i$ we use a subset of S_i^{in} by sampling K random walks (without repetition) starting from $\mathbf{v}_i$, such that each random walk visits at most L different nodes. We find this random sampling to lead to a more robust model as it learns to generate from incomplete and diverse input representations. Optionally, we can also feed disconnected adjacent nodes as additional input. We found this to perform similarly in the task of road modeling due to high connectivity in data.

Decoder

We decode the outgoing nodes V_i^{out} with a decoder GRU. The recurrent structure of the decoder enables capturing local dependencies between roads such as orthogonality at road intersections. We independently predict $\Delta x_{t'}^{out}$ and $\Delta y_{t'}^{out}$ for an outgoing node $\mathbf{v}_{t'}^{out}$, indicating a new node's relative location w.r.t. $\mathbf{v}_i$. Additionally, we predict a binary variable which indicates whether another node should be generated. At generation time we check overlap between the new node and existing graph with a $5m$ threshold to produce loops. Optionally, we predict the edge type between $(\mathbf{v}_i, \mathbf{v}_{t'}^{out})$, i.e. minor or major road, using a categorical variable. The hidden state $\mathbf{h}_{t'}$ of the decoder is updated as:

$$\mathbf{h}_{t'+1} = \text{GRU}(\mathbf{h}_{t'} \mid \mathbf{h}_{\text{enc}}, \mathbf{h}_{\text{attr}}, \Delta\mathbf{x}_{t'}^{out}) \tag{3.1}$$

3.4.3 Aerial Road Parsing

Parsing with Map Prior

The dominant approaches to parse roads from aerial imagery have trained CNNs to produce a probability (likelihood) map, followed by thresholding and thinning. This approach typically results in maps with

holes or false positive road segments, and heuristics are applied to postprocess the topology. We view a NTG model trained to generate city graphs (i.e. *not trained* for road parsing) as a prior, and use it to postprocess the likelihood coming from the CNN. Starting from the most confident intersection node as the root node, we push all its neighbors into the queue of unvisited nodes, and then use NTG to expand the graph. At each decoding step, we multiply the likelihood from CNN with the prior produced by NTG, and sample output nodes from this joint distribution. The end of sequence is simply determined by checking whether the maximum probability of a new node falls below a threshold (0.05 in our paper).

Image-based NTG

We also explore *explicitly training* NTG for aerial parsing. We condition on image features by including predictions from a CNN trained to parse aerial images in the attribute vector $\mathbf{h}_{attr}$. In practice, we initialize the graph obtained by thresholding and thinning the ouputs of the CNN, and use the trained image-based NTG on top.

3.4.4 Implementation Details

We exploit the same NTG model in both tasks of city generation and road detection. Depending on the task, we empirically find that the best parameterization strategy varies. For city generation, we use discrete Δx, Δy with resolution of $1m$ for both encoder and decoder, where x points to east and y points to north. Here, Δx and Δy are limited to $[-100 : 100]$, indicating that the largest offset in either direction is $100m$. The discrete Δx and Δy values are given associated embeddings (resembling words in language models), which are concatenated to generate the input to the encoder at every step. For road detection, we use continuous polar coordinates in the encoder, where the axis is rotated to align with the edge from the previous to the current node. This forms rotation invariant motion trajectories that help detecting roads with arbitrary orientation. The decoder always uses a discrete representation. We encode and decode the coordinates x and y independently. We find that this yields similar results compared to joint prediction, while significantly saving training memory and model capacity. 500 hidden units are used in encoder and decoder GRUs.

3.4.5 Learning & Inference

Inference

At each inference step, we pop a node from the queue Q, encode its existing incoming paths, and generate a set of new nodes. For each new node, we check if it is in the close proximity of an existing node in the graph. If the distance to an existing node is below a threshold ϵ ($5m$ in our paper), we do not add the new node to the queue. Instead, an edge is included to connect the current node to the existing node. This enables the generation of cycles in the graph. We also find the maximum node degree, maximum node density, and minimum angle between two edges in the training set, and ensure our generation does not exceed these limits. We refer to supplemental material for their effects.

Learning

At training time, K incoming paths for each $\mathbf{v}_i$ are sampled, and we learn to predict *all* of its neighboring nodes. We enforce an order in decoding the nodes, where we sort nodes counter-clockwise to form a

	Country	City	Node	Edge	Area	Length
RoadNet	13	17	$233.6k$	$262.1k$	$170.0km^2$	$7410.7km$
SpaceNet	4	4	$115.8k$	$106.9k$	$122.3km^2$	$2058.4km$

Table 3.1: Dataset statistics of RoadNet and SpaceNet [2].

Method \ Metric	*Perceptual*					*Urban Planning*					*Diversity*
	`mp1` 10^{-1}	`mp2` 10^{0}	`pa` 10^{-1}	`fc` 10^{1}	`rate`	`densi.` 10^{1}	`conne.` 10^{-2}	`reach` 10^{5}	`conve.` 10^{-3}	`rate`	
GraphRNN-2D [11, 131]	7.12	6.35	8.45	16.15	25.0	51.58	4.61	45.11	6.72	43.7	44.26
PGGAN [52]	1.98	2.15	5.34	10.51	63.2	45.77	19.48	4.33	2.94	58.9	<u>5.95</u>
CityEngine-5k [1]	2.74	2.71	8.34	14.78	47.1	13.59	21.66	7.61	16.66	51.7	45.86
CityEngine-10k [1]	2.55	2.56	8.23	14.17	48.9	12.43	21.79	7.05	16.82	52.1	46.00
NTG-vanilla	2.63	2.33	4.05	9.17	66.0	8.69	**1.87**	8.99	3.06	86.5	41.27
NTG-enhance	**1.52**	**1.34**	**2.83**	**6.76**	**77.3**	**3.76**	1.97	**4.13**	**1.86**	**92.4**	42.09

Table 3.2: Perceptual domain-adapted FIDs ({maxpool1,maxpool2,pre-aux,fc}, lower is better), Urban Planning feature differences ({density,connectivity,reach,convenience}, lower is better), and Diversity evaluation of city generation. Ratings (higher is better) are computed by averaging with scales {10,10,10,20} for perceptual and {60,30,50,20} for urban planning. <u>Extremely low</u> Diversity indicates incapability of creating new cities.

sequence. The ordering saves having to solve an assignment problem to compute the loss function. Our model is trained using ground truth map data with teacher-forcing [128], using a cross entropy loss for each of the output nodes. The networks are optimized using Adam [53] with a learning rate of 1e-3, weight decay of 1e-4, and gradient clipping of 1.0.

3.5 Experiments

We demonstrate NTG on three tasks: city road layout generation, satellite road parsing, and environment simulation.

3.5.1 City Road Layout Generation

RoadNet Dataset

We collected a real-world road dataset from OpenStreetMap (OSM) to facilitate this task. In particular, we selected 17 unique cities across continents and gathered all road markers. OSM, being crowd-sourced, often has incomplete markers in underpopulated areas. To alleviate this, we manually select the most densely annotated $10km^2$ region within each city. These constitute our final RoadNet dataset. Table 3.1 shows the statistics.

Metrics

The goals of road layout generation are to create road networks that are: **a)** Perceptually plausible and preserve a meaningful city style, and **b)** Diverse. We use three broad categories of automatic metrics to evaluate city generation:

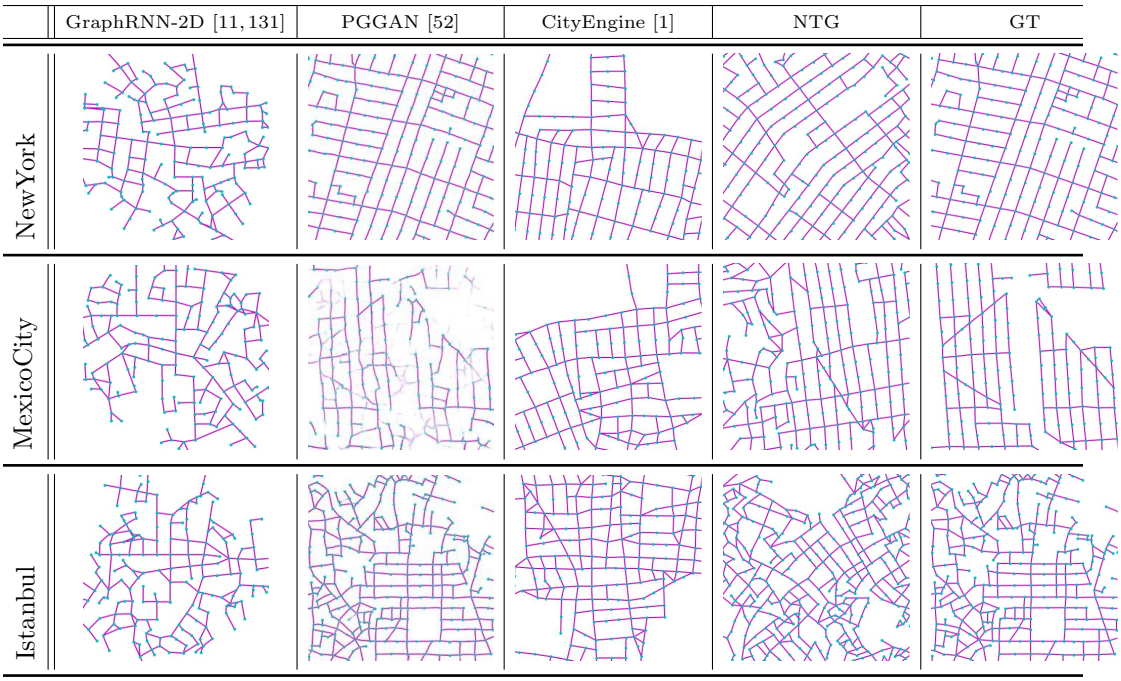


Figure 3.3: Qualitative examples of city road layout generation. GraphRNN-2D generates unnatural structures and fails to capture city style. PGGAN is unable to create new cities by either severely overfitting, or producing artifacts. CityEngine produces less style richness due to its fixed rule-based synthesis algorithm. NTG is able to both capture the city style and creating new cities.

Perceptual: For every node, we render the graph in a $300m$ neighborhood centered around it on a canvas. With their perceptual features extracted from an InceptionV3 network [108], we compute the Fréchet Inception Distance (FID) [45] between the synthesized roads and ground truth maps for each city. To ensure a meaningful FID, we adapt InceptionV3, which has originally been trained on natural images, to road drawings, by finetuning it to predict city ids on our dataset. This yields a 90.27% accuracy, indicating effective capture of style across cities in the network.

Urban Planning [6]: We measure four common urban planning features reflecting city style: **1.** Node *density* within a neighborhood of $100m$, $200m$, and $300m$. **2.** *Connectivity* as reflected by the degrees of nodes. **3.** *Reach* as the total length of accessible roads within a distance of $100m$, $200m$, and $300m$. **4.)** Transportation *convenience* as the ratio of the euclidean distance over the Dijkstra shortest path for node pairs that are more than $500m$ away. We also compute the Fréchet distance between normal distributions computed from the concatenation of the Urban Planning features of real and generated maps.

Diversity metric: We measure the ability to create novel cities by computing the overlap between a real and generated city as the percentage of road in one graph falling outside the $10m$ vicinity of the road in the other graph, and vice versa. We compare this Chamfer-like distance against all ground truth maps and report the average lowest value.

Results

We compare the following methods:

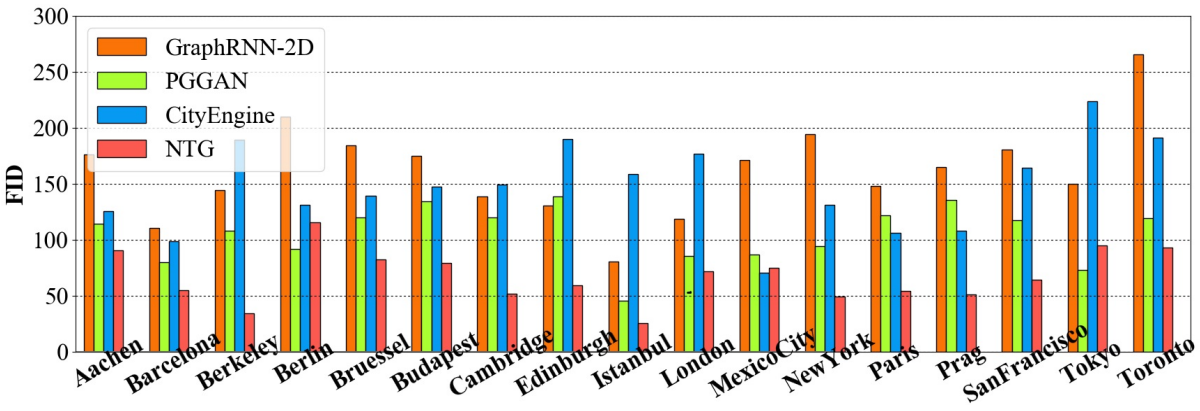


Figure 3.4: City-wise FID (fc) of different methods.

• **GraphRNN-2D [11, 131]**: We enhance the GraphRNN model by introducing extra branches to encode/decode node coordinates and city id. We add a CNN that takes into account local rendering of existing graph as in [11] and add checks to avoid invalid edge crossing during inference.

• **PGGAN [52]**: We train to generate images of road layouts at a resolution of 256×256. We use our trained InceptionV3 network to classify samples into cities to compute city-wise metrics. For computing the graph-related metrics we convert images to graphs by thresholding and thinning.

• **CityEngine [1]**: CityEngine is a state-of-the-art software for synthesizing cities based on an optimized, complex rule-based L-system [83]. By default, it only offers limited templates and is incapable of generating new cities. To enhance CityEngine, we use its provided control interface and exhaustively search over its attribute space by enumerating combinations of important control parameters such as angles, bending specifications, and crossing ratios. We then predict city probabilities using the InceptionV3 network, and select the highest ranking $10km^2$ as the result for each city.

• **NTG**: NTG begins with a root node with its edges. We evaluate NTG with a random root (NTG-vanilla), as well as with a pre-stored high connectivity root (NTG-enhance).

Tab. 3.2 and Fig. 3.3 show quantitative and qualitative results, respectively. Quantitatively, NTG outperforms baselines across all metrics. GraphRNN-2D fails to capture various city styles and frequently produces unnatural structures. This is due to its sequential generative nature that depends on Breadth First Search. The RNN that encodes the full history fails to capture coherent structures since consecutive nodes may not be spatially close due to BFS. PGGAN [52] produces sharp images with occasional artifacts that are difficult to convert into meaningful graphs. Samples from PGGAN are severely overfit as reflected by the Diversity metric, indicating its inability to create new cities. Moreover, PGGAN also suffers from mode-collapse and memorizes a portion of data. This imbalance of style distribution leads to worse perceptual FIDs. With our enhancement (exhaustive search), CityEngine [1] is able to capture certain cities' style elements: especially the node density. However, it has less topological variance and style richness due to its fixed set of rules. Expanding its search from $5000km^2$ (CityEngine-5k) to $10000km^2$ (CityEngine-10k) of generated maps does not lead to significant improvements, while requiring double the amount of computation. NTG is able to create new cities, while better capturing style in most cities as shown in Fig. 3.4.

Fig. 3.5 digs into NTG's generated maps, showing that NTG learns to remember local road patterns. As the graph grows, different local topologies are organically intertwined to produce new cities. Fig. 3.6

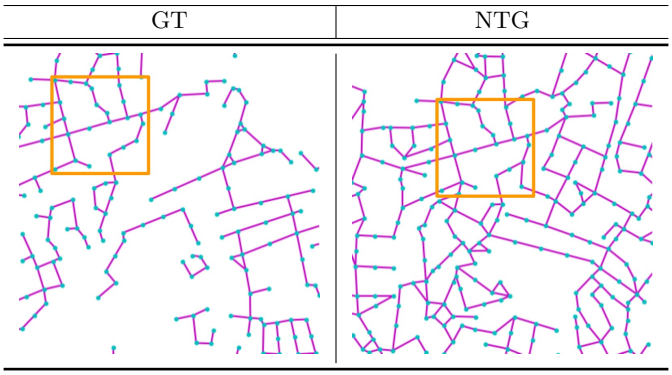


Figure 3.5: NTG creates new city road layouts in a combinatorial manner. Local patterns as shown by orange boxes are remembered, and then intertwined to create novel structures.

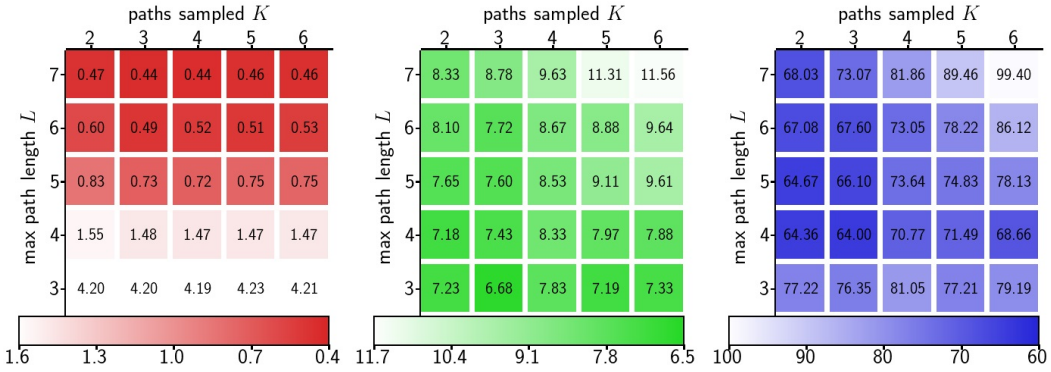


Figure 3.6: Effect of sampled paths K and maximum path length L on reconstruction quality in meters (red), inference time in seconds per km^2 (green), and FID-fc (blue).

shows the effect of two important hyper-parameters: maximum number of paths K and maximum incoming path length L. Results show that reconstruction quality is determined by L, while K and L both affect inference time. Training with longer and more paths does not necessarily improve perceptual quality, since generation starts from a single root node without long paths.

We further demonstrate our approach by having NTG predict two types of roads, i.e. major and minor roads. Results are shown in Fig. 3.8, showing that NTG easily generalizes to a more complex modeling problem.

Interactive Synthesis

We showcase an application for interactive road layout generation where a user chooses from a palette of cities and provides local topology priors by sketching. We match the user's input with pre-stored node templates to form the root node. To allow generating multiple components on the same canvas, we simply modify the NTG inference procedure to iterate through multiple queues in parallel. Fig. 3.7 shows examples of the generation process.

Beyond Road Layouts

In Appendix, we show results on using NTG's multipath paradigm for learning effective representation of complex shapes, such as multi-stroke hand drawings. This shows potential as a general purpose spatial graph generative model beyond the city road modeling.

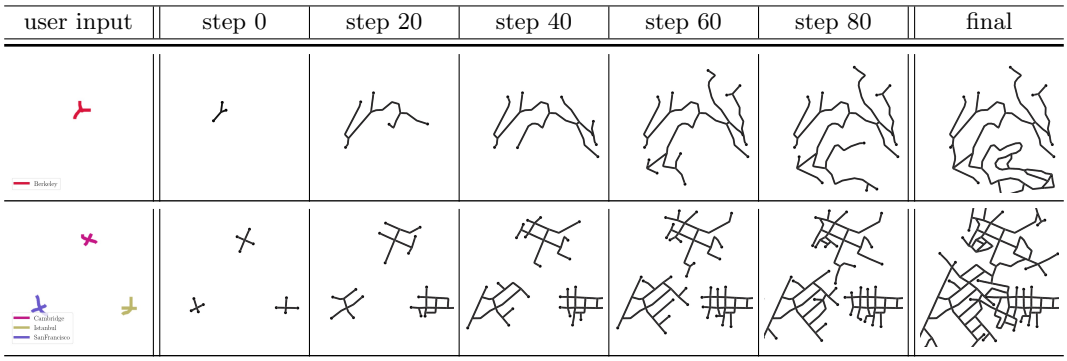


Figure 3.7: Examples of interactive city road layout generation via user sketching and local style selection.

3.5.2 Satellite Road Parsing

SpaceNet Dataset

While several datasets have been presented for road detection [2, 11, 31, 121], we use SpaceNet [2] for its large scale, image quality, and open license. To facilitate consistent future benchmarking, we reformat the raw data into an easy-to-use version with consistent tile size in metric space. Tab. 3.1 shows its statistics. We split tiles of each city into train-validation-test with a 4-1-1 ratio.

Metrics

Average Path Length Similarity (APLS) has been shown to be the best metric to reflect routing properties [2]. Between two graphs, APLS is defined as

$$\text{APLS} = 1 - \frac{1}{N_p} \sum_{p_{\mathbf{v}_1\mathbf{v}_2} < \infty} min \left\{ 1, \frac{|p_{\mathbf{v}_1\mathbf{v}_2} - p_{\mathbf{v}'_1\mathbf{v}'_2}|}{p_{\mathbf{v}_1\mathbf{v}_2}} \right\}$$

where $\mathbf{v}$ denotes a source graph node, $\mathbf{v}'$ as its closest on-road point in the target graph if such a point exists within a buffer range ($5m$), N_p number of paths. Here, $p_{\mathbf{v}_1\mathbf{v}_2}$ denotes the Dijkstra shortest path length between two nodes, and has infinite value if no path exists. We also exchange source and target graphs to establish metric symmetry. To ensure even node distribution, graphs are RDP-simplified [34,86] and uniformly subdivided with $30m$ maximum edge length. While we use APLS as our main metric, we also report conventional pixel-wise IOU and F1 score as references, even though they are less desirable as revealed in [2].

Results

We compare three categories of methods:

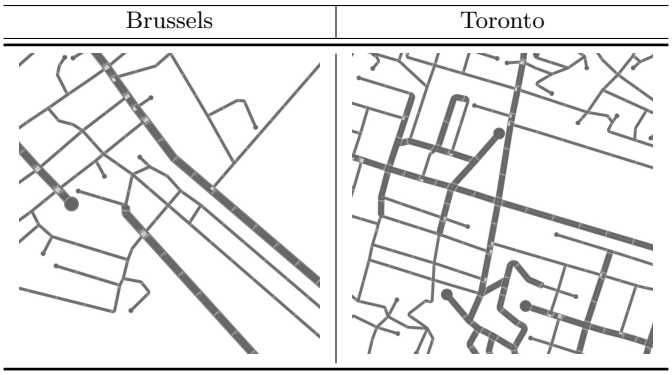


Figure 3.8: NTG can be easily extended to generate road type.

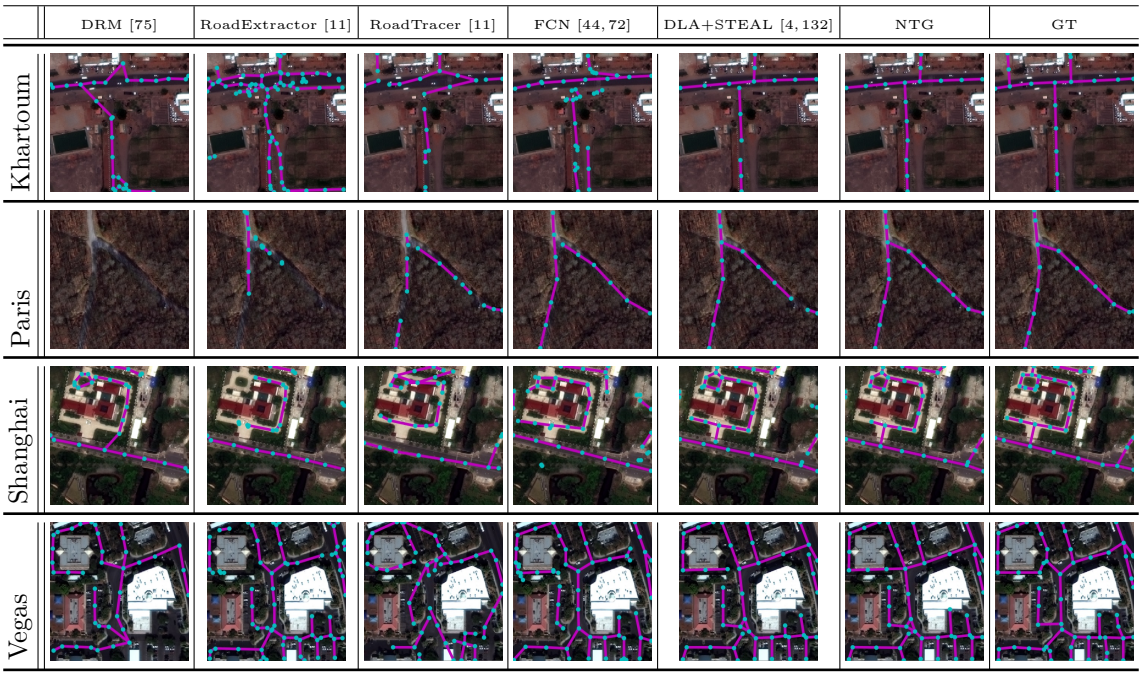


Figure 3.9: Qualitative examples of SpaceNet road parsing.

• **Prior art**: We evaluate DeepRoadMapper [75], RoadExtractor [11], and RoadTracer [11]. RoadTracer requires additional starting points as input. We use the most likely pixel predicted by their CNN, as well as 30 points randomly selected from ground truth (RoadTracer-30).

• **Stronger CNNs**: We explore more powerful CNN architectures. We train an FCN with a ResNet backbone [44, 72], as well as a CNN using DLA [132] with STEAL [4]. To obtain the graph we use standard thinning and geodesic sorting.

• **NTG**: We evaluate both the parsing NTG (NTG-P) that is only trained on RoadNet and acts as a topological prior and image-based NTG (NTG-I) that is trained on SpaceNet.

Table 3.3 and Figure 3.9 present SpaceNet results. It can be seen that our method outperforms baselines in all metrics. The DLA+STEAL CNN produces cleaner predictions that focus on road. NTG-P trained only with RoadNet is able to successfully parse graph structure. Using NTG-I that

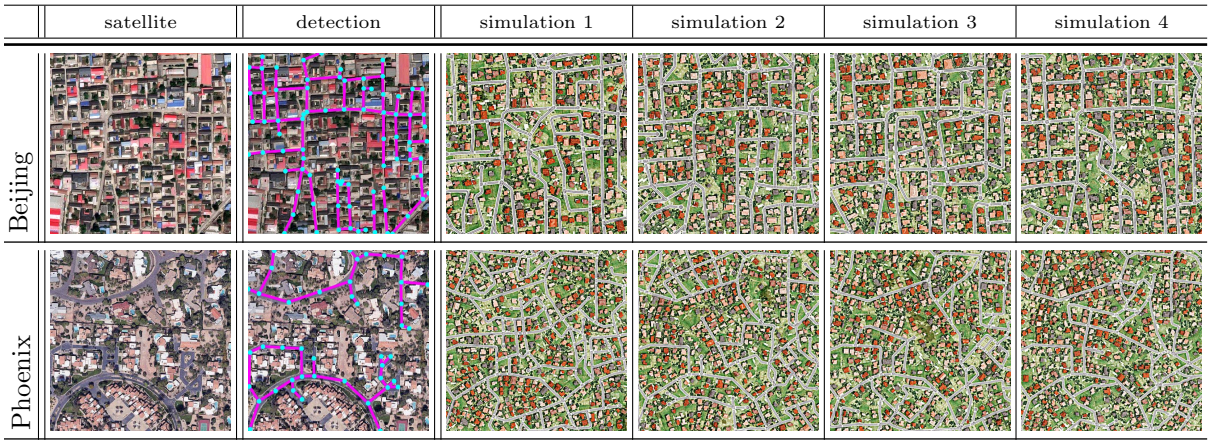


Figure 3.10: Sat2Sim: converting satellite image into a series of simulated environments. Buildings and vegetation added via [1].

	IOU	F1	APLS
DeepRoadMapper [75]	45.02	62.08	51.49
RoadExtractor [11]	52.91	69.20	57.38
RoadTracer [11]	10.23	18.56	48.55
RoadTracer-30 [11]	48.29	65.13	42.94
FCN [44, 72]	51.09	67.63	56.56
DLA+STEAL [4, 132]	58.96	74.18	71.04
NTG-P ([75]'s CNN)	50.58	67.18	55.87
NTG-P ([11]'s CNN)	51.62	68.09	58.79
NTG-P (DLA+STEAL)	59.29	74.44	70.99
NTG-I (DLA+STEAL)	**63.15**	**77.42**	**74.97**

Table 3.3: Comparison of methods on the standard SpaceNet split.

	IOU	F1	APLS
RoadExtractor [11]	20.51	34.03	43.06
DLA+STEAL [4, 132]	33.94	50.68	56.15
NTG-P (DLA+STEAL)	**35.16**	**52.02**	**57.89**

Table 3.4: SpaceNet evaluation on unseen city by holding one city out in training. Without finetuning, the RoadNet pretrained NTG-P is able to improve over DLA+STEAL.

further takes CNN output as input achieves the best result. We also experiment the RoadNet trained NTG-P with CNNs from prior art [11, 75]. It can be seen that the city topology knowledge of NTG makes it a better graph extractor compared to hand-crafted postprocessings in [11, 75], especially in terms of APLS. For NTG-P with DLA+STEAL we notice it has similar performance as standard graph extraction. This is because DLA+STEAL prediction has high confidence as it is trained and tested with same cities that have similar visual appearance. We therefore further experiment with one city held-out to simulate road parsing in unseen cities. Results are presented in Table 3.4. It can be seen that NTG-P is able to further improve the result, demonstrating the effectiveness of generative road layout knowledge learnt from RoadNet. We conduct 4-fold evaluation holding out each city per fold, and report the average result.

3.5.3 Environment Simulation

We further showcase a novel application that combines our two tasks in Figure 3.10. We propose to directly convert a satellite image into simulation-ready environments, which may be important for testing autonomous vehicles in the future. First, we detect roads in the satellite image with NTG, giving us an initial graph. Then, we exploit our generative model to propose plausible variations. This is done by pushing all single-connection nodes in the parsed graph into our generative queue, and running the NTG generative process to expand the graph. We directly make use of the NTG model trained for city generation and choose a random city id for each run. This has two main advantages. First, it is fully automatic and only requires a low-cost satellite image as input. Second, it provides a set of plausible variations of the environment (city) instead of a static one, which could eventually enable training more robust agents. For visualization, we additionally add buildings and tree via [1], showing plausible and diverse simulation-ready cities.

3.6 Conclusion

In this paper, we proposed Neural Turtle Graphics for generating large spatial graphs. NTG takes the form of an encoder-decoder neural network which operates on graphs locally. We showcased NTG on generating plausible new versions of cities, interactive generation of city road layouts, as well as aerial road parsing. Furthermore, we combined the two tasks of aerial parsing and generation, and highlighted NTG to automatically simulate new cities for which it has not seen any part of the map during training time. In future work, we aim to tackle generation of other city elements such as buildings and vegetation.

Chapter 4

A Face-to-Face Neural Conversation Model

4.1 Abstract

Neural networks have recently become good at engaging in dialog. However, current approaches are based solely on verbal text, lacking the richness of a real face-to-face conversation. We propose a neural conversation model that aims to read and generate facial gestures alongside with text. This allows our model to adapt its response based on the "mood" of the conversation. In particular, we introduce an RNN encoder-decoder that exploits the movement of facial muscles, as well as the verbal conversation. The decoder consists of two layers, where the lower layer aims at generating the verbal response and coarse facial expressions, while the second layer fills in the subtle gestures, making the generated output more smooth and natural. We train our neural network by having it "watch" 250 movies. We showcase our joint face-text model in generating more natural conversations through automatic metrics and a human study. We demonstrate an example application with a face-to-face chatting avatar.

4.2 Introduction

We make conversation everyday. We talk to our family, friends, colleagues, and sometimes we also chat with robots. Several online services employ robot agents to direct customers to the service they are looking for. Question-answering systems like Apple Siri and Amazon Alexa have also become a popular accessory. However, while most of these automatic systems feature a human voice, they are far from acting like human beings. They lack in expressivity, and are typically emotionless.

Language alone can often be ambiguous with respect to the person's mood, unless indicative sentiment words are being used. In real life, people make gestures and read other people's gestures when they communicate. Whether someone is smiling, crying, shouting, or frowning when saying "thank you" can indicate various feelings from gratitude to irony. People also form their response depending on such context, not only in what they say but also in how they say it. We aim at developing a more natural conversation model that jointly models text and gestures, in order to act and converse in a more natural way.

Figure 4.1: Facial gestures convey sentiment information. Words have different meanings with different facial gestures. Saying "*Thank you*" with different gestures could either express gratitude, or irony. Therefore, a different response should be triggered.

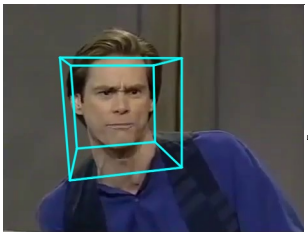
Figure 4.2: Example conversations from our MovieChat dataset. Each row shows two examples, left shows query face and text, right shows target face and text. Our dataset has various conversation scenarios, such as simple conversations shown in the first and second rows on the left, as well as more challenging cases shown on the right.

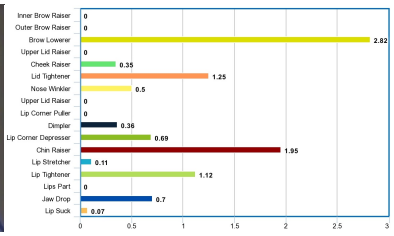
Recently, neural networks have been shown to be good conversationalists [62, 119]. These typically make use of an RNN encoder which represents the history of the verbal conversation and an RNN decoder that generates a response. [63] built on top of this idea with the aim to personalize the model by adapting the conversation to a particular user. However, all these approaches are based solely on text, lacking the richness of a real face-to-face conversation.

In this paper, we introduce a neural conversation model that reads and generates both a verbal response (text) and facial gestures. We exploit movies as a rich resource of such information. Movies show a variety of social situations with diverse emotions, reactions, and topics of conversation, making them well suited for our task. Movies are also multi-modal, allowing us to exploit both visual as well as dialogue information. However, the data itself is also extremely challenging due to many characters that appear on-screen at any given time, as well as large variance in pose, scale, and recording style.

Our model adopts the encoder-decoder architecture and adds gesture information in both the encoder as well as the decoder. We exploit the FACS representation [35] of gestures, which allows us to effectively encode and synthesize facial gestures. Our decoder is composed of two levels, one generating the verbal response as well as coarse gesture information, and another level that fills in the details, making the generated expressions more natural. We train our model using reinforcement learning that exploits a trained discriminator to provide the reward. We show that our model generates more appropriate responses compared to multiple strong baselines, on a large-scale movie dataset. We further showcase NeuralHank, an expression-enabled 3D chatting avatar driven by our proposed model.

The rest of the paper is organized as follows. Sec. 4.3 reviews the related work. In Sec. 4.4 we introduce our dataset to facilitate face-to-face conversation modeling. In Sec. 4.5 we describe our approach.





query	target	# examples
text	text	40,200,261
text+face	text	48,475
text	text+face	48,475
text+face	text+face	24,727

(a) (b) (c)

Figure 4.3: Overview of our MovieChat database. (a) and (b) show an example frame with 3D face detection and detected FACS intensities. We obtain detections using the off-the-shelf OpenFace [10] package. (c) shows the scale of our MovieChat database. Our database is by far the largest language-face conversation video dataset.

Sec. 4.6 provides extensive evaluation and introduces our chit-chatting avatar.

4.3 Related Work

4.3.1 Dialogue Systems

Dialogue systems have been explored since the 60', with systems like ELIZA [126] and PARRY [27] already capable of engaging in relatively complex conversations. These approaches have mainly been based on hand-coded rules, thus were not able to adapt to users and topics, and usually seemed unnatural. In [90], the authors formulated the problem as statistical machine translation, where the goal was to "translate" the query posts in blogs into a response. This problem setting is typically harder than traditional translation from one language to another, since the space of possible responses is more diverse.

4.3.2 Neural Conversation Models

Conversation modeling has recently been gaining interest due to the powerful language models learned by neural networks [62,63,119]. [119] was the first to propose a neural conversation model, which exploited the encoder-decoder architecture. An LSTM encoder was used to represent the query sentence while the decoder LSTM generated a response, one word at a time. The model was trained on a large corpus of movie subtitles, by using each sentence as a query and the following sentence as a target during training. Qualitative results showed that meaningful responses were formed for a variety of queries. In parallel, the Skip-Thought model [55,138] adopted a similar architecture, and was demonstrated to be effective in a variety of NLP tasks as well as image-based story-telling.

Since neural conversation models typically produce short and more generic sentences, the authors in [62] proposed an improved objective function that encouraged diversity in the generator. In [96], the authors exploited a hierarchical encoder-decoder, where one GRU layer was used to model the history of the conversation at the sentence level, and the second level GRU was responsible for modeling each sentence at the word level. This model was extended in [97] by adding latent variables aiming to capture different topics of conversation, allowing the model to achieve a higher diversity in its response.

An interesting extension was proposed in [63] which aimed at personalizing conversations. The model

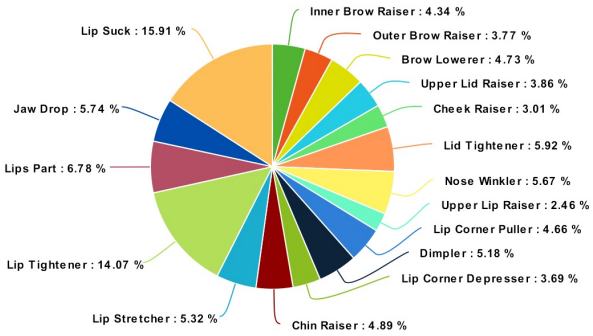


Figure 4.4: List of gestures recorded in the MovieChat dataset, and percentage of frames where each gesture is dominant.

learned a separate embedding for each person conversationalist, jointly with dialogue. The purpose of the embedding was to bias the decoder when generating the response. This allowed for a more natural human-like chit-chat, where the model was able to adapt to the person it was speaking to.

4.3.3 Face and Gesture Modeling

Most of these works are based solely on language. However, humans often use body gestures as an additional means to convey information in a conversation. An interesting approach was proposed in [59, 60] which aimed at synthesizing body language animations conditioned on speech using a HMM. This approach required motion capture data recorded during several conversation sessions.

Face capture has been a long-studied problem in computer vision, with many sophisticated methods such as [10,49,99]. The FirstImpression dataset [13] was collected to facilitate the need of data in gesture recognition. Face synthesis has been widely studied in both vision and graphics communities. [105] proposed a reconstruction algorithm that captures a person's physical appearance and persona behavior. [114] transfers facial gesture from a source video to a target video to achieve realistic reenactment. [106] further transformed audio speech signal into a talking avatar using an RNN-based model.

In our approach, we aim to both encode and generate facial gestures jointly with language, by exploiting a large corpora of movies. Movies feature diverse conversations and interactions, and allow us to use both visual as well as dialogue information.

4.4 The MovieChat Dataset

Datasets of considerable size are key to successfully training neural networks. In our work, we seek a dataset containing people engaging in diverse conversations, that contains both video as well as transcribed dialogues.

Towards this goal, we build the MovieChat dataset. We take advantage of the large movie collection of MovieQA [109], which contains clips from 250 movies, covering more than half of each movie in duration. To track 3D faces and detect facial gestures, we use the off-the-shelf OpenFace [10] package. Tracking and detection runs in real-time while maintaining good accuracy. This makes processing of such a large volume of video data possible.

However, even the best automatic face detector occasionally fails. Certain recording styles, such as the shaky and free-cam clips, make our processing more challenging. To address these problems

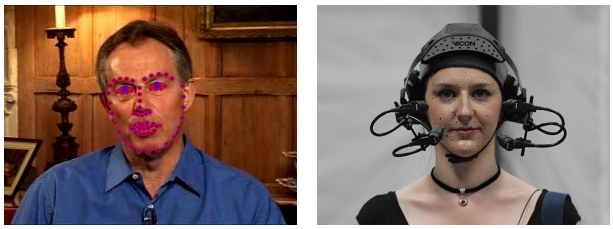


Figure 4.5: Facial Landmark (FL) systems. Left shows an example of "in the wild" landmarks [10,28,137], which fails to capture subtle gesture information. Right shows invasive motion capture landmarks [3].

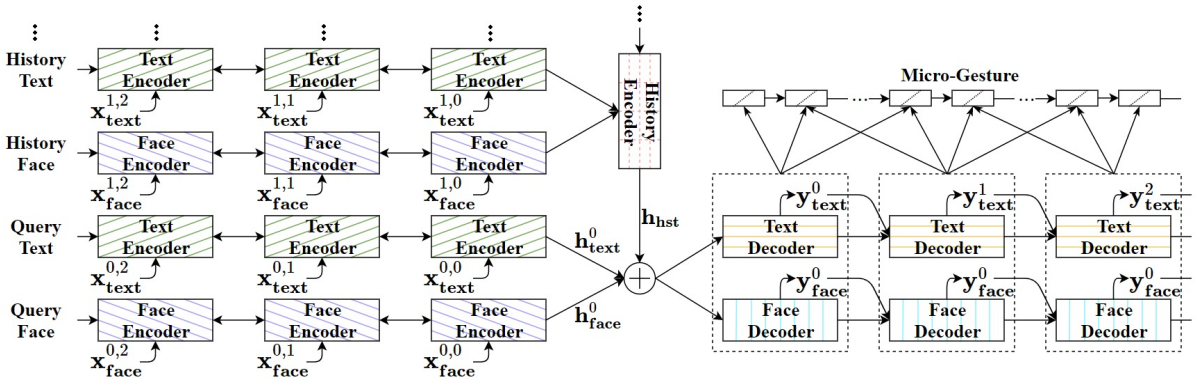


Figure 4.6: Our face-to-face conversation model. Our model consists of 6 RNNs shown in different colors. First, the text-face sequences of query and conversation history are encoded by text and face encoders (only one history sentence is depicted). History sentence encodings are further encoded by the history encoder. Next, encodings are added to form the context vector, which the text and face decoders are conditioned on. Finally, we generate frame-level, micro-gesture animation controls based on the word decodings.

and improve the quality of our dataset, we further divide all movies into short, single sentence clips by exploiting the time stamps stored in their subtitles file. We only keep clips where a single face is detected across all of its frames, and discard the rest of the clips. This is to avoid ambiguous dialog-face association when multiple characters appear in a single shot. Additionally, we remove fast-cut clips where the speaker's face is not fully visible throughout the clip. Finally, we also remove clips in which tracks are extremely shaky, which often suggests tracking failure. We observe significant quality improvement after these filtering steps, with only rare failure cases.

We build our final dataset with the remaining clips. We record image frames, time stamps, 3D face poses, facial gestures, and transcribed dialogues. Fig. 4.3 shows an example, and provides statistics summarizing our dataset. Fig. 4.4 shows the recorded gestures and their statistics in our MovieChat data.

4.5 Face-to-Face Neural Conversation Model

We first explain our facial gestures representation using Facial Action Coding System (FACS) [35]. We then describe our proposed model.

4.5.1 FACS Gesture Representation

Various approaches are available for representing gesture numerically, e.g. Six Universal Expressions (SUE) [14], Facial Landmarks (FL) [28, 137], and FACS [35].

SUE [14] categorizes gesture into six emotions: anger, disgust, fear, happiness, sadness and surprise. It is effective in encoding high-level emotion, but it is overly abstract to describe detailed gestures. Each emotion involves a combination of up to 6 muscle movements, making it difficult for face synthesis and animation.

FL [28, 137] represents gesture using landmark points. Typically, 68 points are used to track corner-edge keypoint positions of the face. Compared to SUE, FL carries more details. However, FL has two disadvantanges. First, FL does not contain complete gesture information. The cheek and forehead regions, which are texture-less but contain many muscles, are usually missing. Second, FL is anatomically redundant. 5 landmarks are used to outline one brow, while its underlying motion is lower dimensional that involves 2 muscle intensity values. Therefore, FL is less desirable for our task. It should be noted there are variations of FL that places landmarks across all muscles evenly. They are widely used in motion capture, e.g. Cara [3] in Figure 4.5. This FL system requires visible marks on the character face, thus making large scale data collection difficult.

We adopt FACS [35] in this paper. Particularly, we use 18 action unit each controls a face muscle, as well as 3 dimensions to represent the 3D head pose. Compared to the SUE and FL, FACS not only captures subtle detail gestures, but also produces highly interpretable gesture representation which makes animation simple and straight-forward. We detect FACS from images using the off-the-shelf OpenFace software [10].

4.5.2 Face-to-Face Conversation Model

Following previous work on conversation modeling [97,104], we adopt the RNN encoder-decoder architecture, but adapt it to our face-to-face conversation task. Our proposed model consists of 6 RNN modules that capture and generate information across different modalities and resolutions. Fig. 4.6 provides an overview of the model. Overall, our model is an encoder-decoder framework that is trained with RL and GAN.

Notation

Our algorithm takes a series of paired text and gesture sequences as input. These represent the query sequence as well as a recent conversation history of N sequences. Here, let $\mathbf{x}^0$ denote the current query sequence, while $\mathbf{x}^n$ indexes the n-th sequence of the current history. We will use subscripts **text** and **face** to denote the data from the two modalities.

Sentence Encoders

We synchronize text and gestures at the word level. Let $\mathbf{x}^{n,\ell}_{\mathbf{text}}$ represent a one-hot encoding of the ℓ-th word in the n-th sentence. To keep the representation consistent and to simplify the multi-dimensional gesture data, we set $\mathbf{x}^{n,\ell}_{\mathbf{face}}$ as a similar one-hot encoding of the closest gesture template. Gesture templates are obtained via k-means ($k = 200$) clustering of all gestures in the training set. We define our sentence-

		beam=1			beam=3			beam=5		
	perp.	*pre. %*	*rec. %*	*F*1	*pre. %*	*rec. %*	*F*1	*pre. %*	*rec. %*	*F*1
Text [55, 104]	32.53	23.18	15.58	17.12	25.00	17.13	18.62	24.70	16.91	18.34
Text+RandFace	32.65	22.92	15.99	17.27	24.74	17.32	18.57	24.71	17.82	18.84
Text+Face	30.17	24.25	17.52	18.69	24.78	18.60	19.40	24.34	18.74	19.37
History-RNN [96]	31.15	23.99	19.46	19.59	23.79	20.11	19.67	23.37	20.50	19.68
History-FC	30.39	24.49	19.61	19.88	24.38	20.50	20.14	23.70	20.45	19.91
Ours-MLE	30.08	25.16	19.72	20.17	24.50	20.32	20.11	23.75	20.47	19.89
Ours-F1	31.91	25.16	20.24	20.42	24.48	20.26	20.02	24.06	20.33	19.96
Ours-GAN	31.60	25.23	20.19	20.44	24.56	20.31	20.08	24.11	20.38	19.97

Table 4.1: The mind-reading text results on text. Second column lists word *perplexity* (lower the better). Third to last columns list unigram *precision*, *recall*, and *F*1-*score* (higher the better) across different beam search size. For each column, we mark the **best** and **second best** results in red and blue color. We underscore the **overall best** result across all methods and all beam sizes.

level encoders as bidirectional RNNs, i.e.

$$\begin{aligned} \mathbf{h}^n_{\mathbf{text}} &= \text{BiLSTM}(\{\mathbf{x}^{n,\ell}_{\mathbf{text}}\}_\ell) \\ \mathbf{h}^n_{\mathbf{face}} &= \text{BiLSTM}(\{\mathbf{x}^{n,\ell}_{\mathbf{face}}\}_\ell) \end{aligned} \tag{4.1}$$

where the BiLSTM computes the forward and backward sentence encodings $\vec{h}^n$ and $\overleftarrow{h}^n$, respectively, concatenates them, and applies a linear layer on top.

History Encoder

To model the context of the conversation, we take a history of N sequences (excluding query), and add another bidirectional RNN over the encoded text and gesture sequences:

$$\mathbf{h}_{\mathbf{hst}} = \text{BiLSTM}(\{\mathbf{h}^n_{\mathbf{text}} \oplus \mathbf{h}^n_{\mathbf{face}}\}_n) \tag{4.2}$$

where $\oplus$ denotes vector concatenation.

Sentence Decoders

We use two decoders that generate the target text and gesture sequences. The output follows the same one-hot encoding as query and history sentences. We condition the target sentence generation on the joint text-face-history context vector, which is obtained by the summation of the query text encoding, query face encoding, and history encoding. Concretely,

$$\mathbf{h}_{\mathbf{enc}} = \mathbf{h}^0_{\mathbf{text}} + \mathbf{h}^0_{\mathbf{face}} + \mathbf{h}_{\mathbf{hst}} \tag{4.3}$$

where $\mathbf{h}_{\mathbf{enc}}$ is the final encoding that we condition our generation decoders on. We use two independent single-directional RNN decoders, i.e.

$$\begin{aligned} \mathbf{h}^\ell_{\mathbf{dec}} &= \text{LSTM}(\mathbf{h}^{\ell-1}_{\mathbf{dec}} \mid \mathbf{h}_{\mathbf{enc}}, \mathbf{y}^{\ell-1}) \\ \mathbf{y}^\ell &= \underset{\mathbf{y}}{\text{argmax}}\ p(\mathbf{y} \mid \mathbf{h}^\ell_{\mathbf{dec}}), \end{aligned} \tag{4.4}$$

where $\mathbf{y}^\ell$ is either ℓ-th output word or gesture, and p computes a softmax over a linear layer on top of the hidden state.

		beam=1			beam=3			beam=5		
	perp.	*pre. %*	*rec. %*	*F*1	*pre. %*	*rec. %*	*F*1	*pre. %*	*rec. %*	*F*1
Face [104]	18.98	26.48	9.83	12.96	22.41	8.18	10.82	20.74	7.55	10.02
Face+RandText	18.94	26.63	10.01	13.15	22.54	8.15	10.82	20.20	7.43	9.80
Face+Text	17.20	29.46	10.89	14.41	25.84	9.46	12.57	24.82	9.14	12.15
History-RNN [96]	20.30	20.84	7.33	9.81	20.84	7.33	9.81	20.84	7.33	9.81
History-FC	20.26	20.86	7.35	9.83	20.81	7.33	9.80	20.84	7.33	9.81
Ours-MLE	17.18	35.81	13.74	18.07	30.44	11.43	15.10	28.25	10.58	13.49
Ours-F1	17.20	36.17	13.92	18.28	30.42	11.43	15.09	28.30	10.63	14.06
Ours-GAN	17.19	36.06	13.85	18.20	30.43	11.38	15.05	28.12	10.52	13.92

Table 4.2: The mind-reading test results on gesture. Legend same as Table 4.1.

Micro-Gesture Generator

The output of our gesture decoder is a gesture template (cluster). We observe that although templates are sufficient for representing semantics, they are insufficient for synthesizing vivid, high framerate animations. We use the micro-gesture module to fill in this resolution gap, effectively interpolating between the consecutive discrete gestures, and adding relevant variations to the final gestures. We define this module as a frame-level RNN. For the t-th frame, we synthesize its micro-gesture based on the two most adjacent words, i.e.

$$\mathbf{h}^t_{\mathbf{micro}} = \text{LSTM}(\mathbf{h}^{t-1}_{\mathbf{micro}} \mid \mathbf{y}^{t-\delta}_{\mathbf{text}} \oplus \mathbf{y}^{t-\delta}_{\mathbf{face}}, \mathbf{y}^{t+\delta}_{\mathbf{text}} \oplus \mathbf{y}^{t+\delta}_{\mathbf{face}}) \tag{4.5}$$

where δ denotes the interpolation interval ($t-\delta$ indexes the previous word, and $t+\delta$ the next one). We obtain the frame-level gesture by linearly regressing $\mathbf{h}^t_{\mathbf{micro}}$ to each individual gesture dimension. These gesture values directly control the muscle intensities that drive a 3D avatar.

Policy Gradient Optimization

As typical, we train the model with the cross-entropy loss. However, the default cross-entropy training suffers from exposure bias, as the model is only exposed to ground truth samples during training. For our decoder networks, we alleviate this problem by optimizing directly for the desired metrics using policy gradient optimization. In this setting, the *policy* takes the form of a decoder RNN, and an *action* is a sentence sampled from the policy denoted by $\tilde{\mathbf{y}}$ (for either **text** or **face**). Our goal is to expose the model to more samples of $\tilde{\mathbf{y}}$, and discover a policy to achieve higher *reward* under the metric of choice evaluated at the end of the sequence (e.g. $F1$-score). This is denoted by $\mathbf{R}(\tilde{\mathbf{y}}, \hat{\mathbf{y}}_{\mathbf{gt}})$, where $\hat{\mathbf{y}}_{\mathbf{gt}}$ is the ground truth sequence. The policy gradient (using a single sample) is computed as follows ($\mathbf{h}$ short for $\mathbf{h}_{\mathbf{enc}}$):

$$\nabla J_{\mathbf{pg}}(\theta) = [\mathbf{R}(\tilde{\mathbf{y}}, \hat{\mathbf{y}}_{\mathbf{gt}}) - \mathbf{b}] \nabla_\theta \log p_\theta(\tilde{\mathbf{y}}|\mathbf{h}) \tag{4.6}$$

where $J_{\mathbf{pg}}$ is the objective function, and $\mathbf{b}$ is the baseline to help reduce the variance of the gradients. We follow [89], and compute the baseline by greedy decoding conditioned on the same $\mathbf{h}$. Computing the baseline in this way mimics the inference strategy at test time, thus obtaining positive gradients whenever the sampled sequence scores higher than the current greedy sequence.

For computing the reward, we will exploit standard metrics such as $F1$-score, as well as learned reward functions as explained next.

Adversarial Discriminator Reward

Conversation models suffers from dull responses [64], while diverse dialogues are preferred in practical scenarios. We address this problem by using a sequence GAN [133], following the idea of [29] for captioning. The *generator* is our decoder network, while the *discriminator* is another network that distinguishes whether the resulting sequence is machine-generated (fake) or real. We can formulate this as a minmax problem, i.e.

$$\min_{\theta} \max_{\eta} J_{\mathbf{gan}}(p_\theta, \mathbf{D}_\eta) \tag{4.7}$$

where $\mathbf{D}$ is the discriminator producing probability value $[0, 1]$, η being its parameters. Specifically, the GAN objective is defined as

$$J_{\mathbf{gan}} = \mathbb{E}_{\hat{\mathbf{y}}_{\mathbf{gt}}}[\log \mathbf{D}_\eta(\mathbf{h}, \hat{\mathbf{y}}_{\mathbf{gt}})] + \mathbb{E}_{\tilde{\mathbf{y}} \sim \mathrm{p}_\theta}[\log (1 - \mathbf{D}_\eta(\mathbf{h}, \tilde{\mathbf{y}}))] \tag{4.8}$$

The discriminator is conditioned on $\mathbf{h}$, trying to both learn what good sequences are and their consistency with the query sequences. When training the discriminator we follow [29], to also add mis-matched query-sequence pairs in the discriminator training step to improve the generation's semantic relevance. We use $\mathbf{D}$ to compute the reward for policy-gradient optimization.

Implementation Details

We substantiate all LSTMs with a 1024-d LSTM cell [47] on top of a 512-d embedding layer, followed by a linear layer with hyperbolic tangent non-linearity to compute the final encoding. Our GAN discriminator is implemented as a 3-layer, 512-d MLP that takes sentence encoding and context vectors as inputs.

Nested hierarchical neural networks are difficult to train from scratch in an end-to-end fashion. We observe the same for our model. To train our model successfully, we first pre-train our text and face encoders on single sequence corpora. Then we freeze the encoder modules and use them to generate setence-level encodings, which is used to pre-train our history model. Similarly, we pre-train decoders to make them familiar with the context. After all modules are pre-trained, we jointly finetune the entire network.

To train our decoder, we adopt the MIXER [87] strategy. We initialize the policy network via MLE. Then we gradually anneal MLE steps and blend in RL steps temporally. We keep this process until all time steps are replaced by RL. To train our discriminator, we mix the same ratio of sampled sequences, ground truth sequences, and mis-matched ground truth. In PG training, we observe that balanced positive and negative rewards are also helpful for the training process. In our case, we randomly discard samples until average reward is equal to the baseline reward. We use clipped gradient descent in our pre-training steps, and Adam [53] in all other training steps.

We pre-train our micro-gesture module on the FirstImpression dataset [13], which contains close-up talking videos that allows high precision tracking of micro-gesture. To synchronize words and gestures at the frame level, we perform speech recognition with Bluemix, and force the alignment with existing transcripts with the Smith-Waterman algorithm [103]. We reduce the jittering effect of our final generation using an online Savitzky-Golay filter [93].

source text	source face sequence	true target text	text only	text+face
we went to the hickory stick, we had a drink, two drinks.		and then? and then i went home alone.	we drank a bottle of champagne.	and then i went to bed.
she doesn't know where he is.		i don't know where he is.	i'm sorry.	i don't know where she is.
and he sleeps only one hour a night.		he's a great man.	he sleeps in the same bed.	he's a good man.
a night that marked the opening of a new chapter.		in world history.	for the future.	in the history of the world.
i hope you're not a hothead like sonny.		he's a good kid.	he's got a lot of something.	he's a good kid.
i guess they was worried they wouldn't find a vein in my arm.		what's that number?	what's that?	i don't think so.
oh, he's so cute.		oh, my god.	oh, my god	he's so cute.
can you hear me? i'm still here. scott. stop.		i'm here. scott. stop.	i'm sorry.	what the f*** are you doing here?
so i don't really remember, yeah.		yeah, right. stupid.	yeah, yeah, yeah.	well, you know what? i'm sorry.
i can't feel my legs.		i can't feel my legs.	and i can't breathe.	it's too much.

Figure 4.7: Success and failure cases of using face along with text. Top five rows show successful examples where adding facial gesture information produces sentences closer to the ground truth. Bottom five rows show failure modes, including face detection failure in the sixth row, and detecting another face that does not belong to the speaker in the seventh row.

4.6 Experiments

We evaluate our model through automatic metrics with a "mind-reading" test, and through a human study with a NeuralHank chatting avatar controlled by our model. We randomly split MovieChat into 4:1:1 train-val-test, and keep the split in all experiments.

4.6.1 The Mind-Reading Test

In the first experiment, we evaluate how well the model's generation matches with the ground truth target text and gesture sequences. This reflects the model's ability to produce appropriate, human-like responses. We note that this is an extremely challenging task, particularly for producing fair evaluation. Due to the multi-modal nature of chit-chat conversations, there exists many plausible responses to the same query, and the ground truth only represents one mode among many. Therefore, we refer to this evaluation as the mind-reading test.

We evaluate the model at both word and sentence level. At the word level, we evaluate the *perplexity*, i.e. the likelihood of generating the correct next target word, given the source and correct previous words in the target sequence. This measures coherence of the textual and facial language models. At the sentence level, we evaluate *precision*, *recall*, and $F1$-*score* between the words in generated sentences and ground truth.

We compare our approach to five baselines: **1.)** Text(Face): The classic Seq2Seq [55, 104] method that uses single modality only (either text or face) and only the query sequence (no history); **2.)** Text+Face(Face+Text): Two encoders for both text and face query senquences without history; **3.)** Text+RandFace(Face+RandText): Same model as previous but trained with randomized face(text)

query sentences; **4.)** History-RNN: Modeling conversation history as well as query text(face) using a hierarchical RNN, which is similar to to [96]; **5.)** History-FC: Same as previous but directly connects history sentences to the decoder with fully connected layers. This exploits the potential in conversation history, at the cost of inflexibility to history length N and significantly heavier models. For our model, we compare Ours-MLE, Ours-F1 ($F1$-score as reward), and Ours-GAN. We use beam search with varying sizes for all methods.

From Table 4.1, 4.2, it can be seen that our method achieves the best performance. Due to non-overlapping conversation scenes between data splits, the improvement of our methods is meaningful and generalizable. Therefore, our experiments quantitatively prove the common intuition that seeing the face makes understanding the conversation easier and better, backing up the main argument of this paper. Our base model can be further improved using reinforcement and adversarial training. GANs do not achieve better automatic metric score than directly setting metric reward for PG. However, GANs are able to generate more diverse and interesting responses, as we will later show in Sec. 4.6.2. This finding is in accordance with image captioning [29].

The Role of Gestures

In Table 4.1, Text+Face outperforms only Text, indicating that gesture information helps text understanding. Text+RandFace does not achieve significant improvement despite a slightly better $F1$-score. This verifies the improvement of Text+Face is indeed due to the effectiveness of gesture data, instead of the additional encoder stream. This justifies our argument that gesture information is useful for text understanding. Our method outperforms both History-RNN and History-FC, showing the compatibility between gesture and history information.

The Role of Text

Similarly, from Table 4.2 it can be seen that in understanding and generating face gestures, text information is helpful. This confirms the mutual benefit between both text and gesture information.

The Role of History

In both text and face, History-FC outperforms History-RNN. This indicates that there is still room for further improvement for better history encoders. However, History-RNN remains the preferable option, for its smaller model size and flexibility to varying history length, which is important in practical scenarios. Interestingly, history method outperforms Text+Face in text mind reading, indicating that multiple sentences of text history is more helpful than a query face sequence. It is the opposite in gesture mind reading, indicating that when guessing facial gestures, seeing the source face and react accordingly can be more helpful than knowing a series of text-only history sentences. This can be also partially due to the nature of movie data, where both source and target can be conveyed by the same character.

4.6.2 The NeuralHank Chatbot

Here, we test how our model's performance in the eyes of real human users. We create a virtual chatbot named NeuralHank, that is controlled by our model. This experiment aims to demonstrate the more realistic potential of our model and provides a pilot study towards new applications, e.g. AI assistants and gaming/HCI.

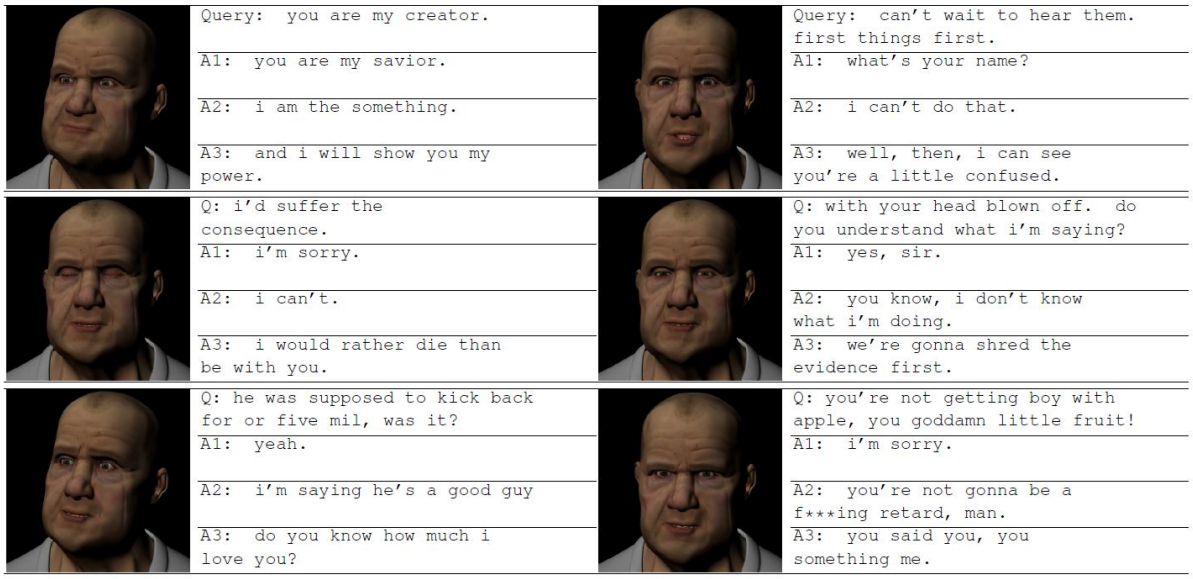


Figure 4.8: NeuralHank examples. `Q` is the query text. `A1`, `A2`, and `A3` are generated by *noMicro-noGAN*, Micro-noGAN, and *Ours*, respectively. We also show one animation frame generated by our method. First two rows show that our GAN-based model generates more diverse and interesting responses. Last row shows failure cases where our method generates confusing responses. Please refer to our project page for animation videos, more examples, and chatting with Hank live through a webcam.

In NeuralHank, we ensemble a series of off-the-shelf packages to convert our model's generation into a real talking avatar. We use Microsoft Speech API to render text as audio, while also keeping record of viseme time tags. We then render FACS gestures using Maya's Facial Animation Toolset, with its default character Hank. For Hank's lip motion, we simply use the viseme event record with tangent interpolation over time. In training, we continue for a few more epochs after the early stopping point until training loss is below a certain threshold. We found this makes the model's generation more particular, which is helpful for building a lively avatar.

We compare three methods: **1.)** *noMicro-noGAN* that uses beam search without GANs, and only word-level face decoder without micro-gesture RNN; **2.)** *Micro-noGAN* that uses the micro-gesture RNN, and sampling without GANs; **3.)** *Ours* as our full model.

We conduct a human study via Amazon Mechanical Turk, by asking participants to rate different methods' responses on the same query. We ask participants to choose the more interesting and natural response, in terms of text, gestures, and overall. For query subjectivity, we randomly choose 65 query sentences from our held-out test set and run all methods using them as inputs. Most participants are not well-trained experts. It is important to make our study easy to follow. To achieve this, we only display a pair of different methods' generations in random order, instead of showing all three together. We also intentionally set query text as the most important information, and set query gesture and history as zero. This makes our task easy to understand, while not affecting the fairness of comparison because the methods only differ on the decoder side.

We request 10 Turkers for each sample. This results in 5850 answers from 37 unique participants. We further use exam questions to filter out the noisy participant responses. The questions are verified samples where one answer is obviously better, e.g. a spot on, grammar error free, and fun sentence, versus a simple and boring yes/no answer.

	text %	*face %*	*overall %*	
noMicro-noGAN	48.8	39.0	46.2	
Micro-noGAN	51.2	61.0	53.8	
noMicro-noGAN	44.8	35.3	42.4	*pairwise*
Ours	55.2	64.7	57.6	
Micro-noGAN	46.1	48.8	46.7	
Ours	53.9	51.2	53.3	
noMicro-noGAN	31.5	25.0	29.8	
Micro-noGAN	32.5	36.8	33.5	*accumu.*
Ours	36.0	38.2	36.6	

Table 4.3: AMT user study on interestingness and naturalness. The evaluation is conducted in form of pairwise comparison. We further accumulate number of votes for different methods.

It can be seen from Table 4.3 that micro-gesture significantly improves gesture quality. Our full model with adversarial training achieves the best user rating from all three perspectives. Compared to no-GAN methods that tend to produce universally correct but less interesting responses, GAN methods produces generally more diverse and interesting responses. However, GAN methods also suffer from occasional confusing or offensive responses.

Fig. 4.8 shows generated samples. We only show one key generated facial gesture per example. Our project page contains videos which better reflect the quality of generations.

4.7 Conclusion

We proposed a face-to-face neural conversation model, an encoder-decoder neural architecture trained with RL and GAN. Our approach used both textual and facial information to generate more appropriate responses for the conversation. We trained our model by exploiting rich video data in form of movies. We evaluated our model through a mind-reading test as well as a virtual chatting avatar. In the future, we aim to learn body controllers as well, model the personalities of the conversation participants, as well as capture more high-level semantics of the situation [118].

Chapter 5

Expressive Telepresence via Modular Codec Avatars

5.1 Abstract

Telepresence in virtual reality (VR) refers to interacting with another human in a virtual space represented by an avatar. Today most avatars are cartoon-like, but soon we will have video-realistic ones. This paper aims in this direction and presents Modular Codec Avatars (MCA), a method to generate hyper-realistic faces driven by the cameras in the VR headset. MCA extends traditional Codec Avatars (CA) by replacing the holistic models with a learned modular representation. It is important to recall that traditional person-specific CAs are learned from few training samples, and typically lack robustness as well as limited expressiveness when transferring facial expressions. MCAs solve these issues by learning a modulated adaptive blending of different facial components as well as an exemplar-based latent alignment. We demonstrate that MCA achieves improved expressiveness and robustness w.r.t to CA in a variety of real-world datasets and practical scenarios. Finally, we showcase new applications in VR telepresence enabled by the proposed model.

5.2 Introduction

Telepresence technologies allow a person to feel as if they were present, to give the appearance of being present, or to have an effect via telerobotics, at a place other than their true location. Telepresence systems can be broadly categorized based on the level of immersiveness. The most basic form of telepresence is video teleconferencing (e.g., Skype, Hangouts, Messenger) that is widely-used, and includes both audio and video transmissions. Recently, a more sophisticated form of telepresence has become available featuring a smart camera that follows a person (e.g., the Portal system).

This paper addresses a more immersive form of telepresence that utilizes a virtual reality (VR) headset (e.g., Oculus, VIVE headsets). VR telepresence aims to enable a telecommunication system that allows remote social interactions more immersive than any prior media. It is not only a key promise of VR, but also has vast potential socioeconomic impact such as increasing communication efficiency, lowering energy footprint, and such a system could be timely for reducing inter-personal disease transmission [46]. VR telepresence has been an important active area of research in computer vision [36, 70, 79, 82, 113, 114, 125].

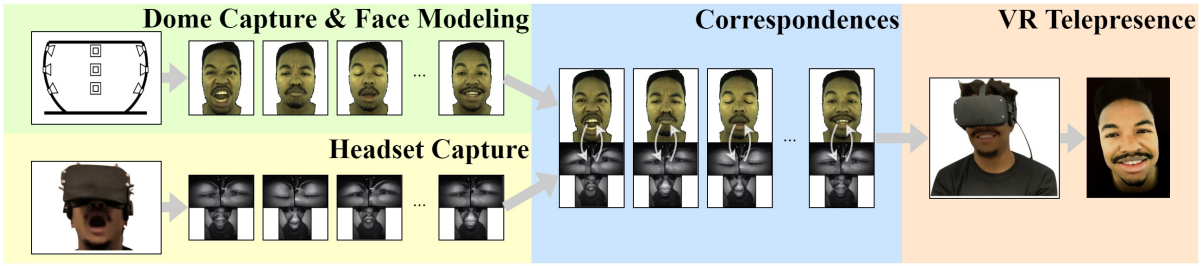


Figure 5.1: Train and test pipeline for our VR telepresence system. In the first stage, we capture facial expressions of a user using both multi-view camera dome and VR headset (mounted with face-looking cameras). Correspondences between VR headset recording and full face expressions are established using the method in [125]. Finally, once the person-specific face animation model is learned using these correspondences, a real-time photo-realistic avatar is driven from the VR headset cameras.

In VR telepresence, the users wear the VR headset, and a 3D face avatar is holographically projected in realtime, as if the user teleports into the virtual space. This allows immersive bidirectional face-to-face conversations, facilitating instant interpersonal communication with high fidelity.

Fig. 5.1 illustrates our VR telepresence system that has three main stages: (1) Appearance/shape capture. The person-specific avatar is built by capturing shape and appearance of the person from a multi-camera system (i.e., the dome). The user performs the same set of scripted facial expressions siting in the dome and wearing a headset mounted with face-looking cameras respectively. The 3D faces are reconstructed from the dome-captured multi-view images, and a variational autoencoder (VAE) is trained to model the 3D shape and appearance variability of the person's face. This model is referred to as Codec Avatar (CA) [70, 125], since it decodes the 3D face from low-dimensional code. The CA is learned from 10k-14k 3D shapes and texture images. (2) Learning the correspondence between the infra-red (IR) VR cameras and the codec avatar. In the second stage, the CA method establishes the correspondence between the headset cameras and the 3D face model using a image-based synthesis approach [125]. Once a set of IR images (i.e., mouth, eyes) in the VR headset are in correspondence with the CA, we learn a network to map the IR VR headset cameras to the codec avatar codes, that should generalize to unseen expressions. (3) Realtime inference. Given the input images from the VR headset, and the network learned in step two, we can drive a person-specific and photo-realistic face avatar. However, in this stage the CA has to satisfy two properties for authentic interactive VR experience:

- **Expressiveness**: The VR system needs to transfer the subtle expressions of the user. However, there are several challenges: (1) The CA model has been learned from limited training samples of the user (10k-14k), and the system has to precisely interpolate/extrapolate unseen expressions. Recall that is impractical to have a uniform sample of all possible expressions in the training set, because of their long-tail distribution. This requires careful ML algorithms that can learn from few training samples.(2) In addition, CA is an holistic model that typically results in rigid facial expression transfers.

- **Robustness**: To enable VR telepresence at scale in realistic scenarios, CAs have to provide robustness across different sources of variability, that includes: (1) iconic changes in the users' appearance (e.g., beard, makeup), (2) variability in the headset camera position (e.g., head strap), (3) different lighting and background from different room environments, (4) hardware variations within manufacturing specification tolerance (e.g., LED intensity, camera placement).

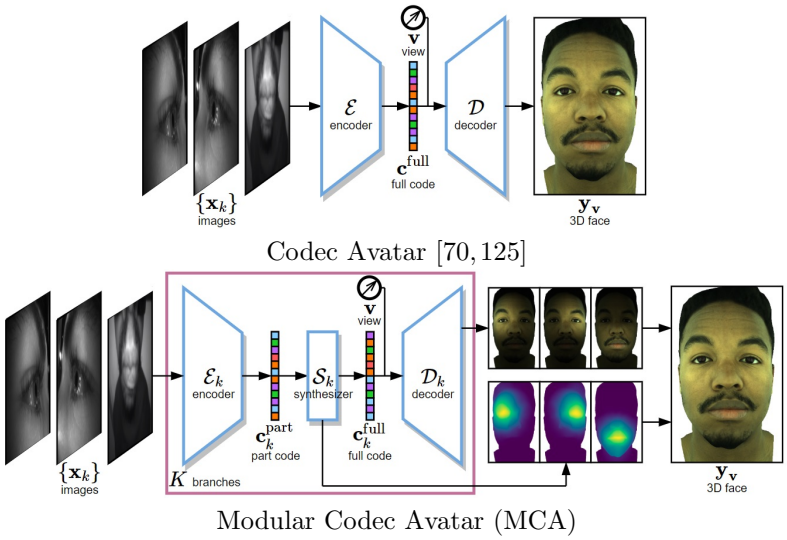


Figure 5.2: Model diagrams comparing the previous CA and the proposed MCA. K denote the number of head-mounted cameras. In CA, images of all headset cameras are feed together to the single encoder $\mathcal{E}$ to compute the full face code which is subsequently decoded into 3D face using deocoder $\mathcal{D}$. In MCA, image of each camera is encoded separately into modular code $\mathbf{c}_k^{\text{part}}$ with the encoder $\mathcal{E}_k$, which is feed to a synthesizer $\mathcal{S}_k$ to estimates a camera specific full face code $\mathbf{c}_k^{\text{full}}$ and blending weights. Finally all these camera specific full face codes are decoded into 3D faces and blended together to form the final full face.

This paper proposes Modular Codec Avatar (MCA) that improves robustness and expressiveness of traditional CA. MCA decomposed the holistic face representation of traditional CA into learned modular local representations. Each local representation corresponds to one headset-mounted camera. MCA learns from data the automatic blending across all the modules. Fig. 5.2 shows a diagram comparing the CA and MCA models. In MCA, a modular encoder first extracts information inside each single headset-mounted camera view. This is followed by a modular synthesizer that estimates a full face expression along with its blending weights from the information extracted within the same modular branch. Finally, multiple estimated 3D faces are aggregated from different modules and blended together to form the final face output.

Our contributions are threefold. First, we present MCA that introduces modularity into CA. Second, we extend MCA to solve the expressivity and robustness issues by learning the blending as well as new constraints in the latent space. Finally, we demostrate MCA's robustness and expressiveness advantages on a large-scale real-world VR dataset.

5.3 Related Work

5.3.1 Morphable 3D Facial Models

Part-based models have been widely used for modeling facial appearance because of their elasticity and the ability to handle occlusion. Cootes et al. [28] introduced Active Appearance Models (AAM) for locating deformable objects such as faces. The facial shape and part-based appearance are iteratively matched to the image, using parameter displacement estimated from residual errors. The 3D morphable

face model by Blanz and Vetter [15] decomposes the face into four parts to augment expressiveness, despite the PCA decomposition is still computed holistically on the whole face. Tena et al. [112] proposed region-based face models that use a collection of PCA sub-models with shared boundaries. Their method achieved semantically meaningful expression bases, while generalising better to unseen data compared to the holistic approach. Neumman et al. [80] extended sparse matrix decomposition theory to face mesh sequence processing, and a new way to ensure spatial locality. Cao et al. [18] proposed to learn part-based rigidity prior from existing facial capture data [20], which was used to impose rigid stability and avoid head pose jittery in real-time animation. Recently, Ghafourzadeh et al. [41] presented local PCA-based model combined with anthropometric measurement to ensure expressiveness and intuitive user control. Our approach also decomposes the face into part-based modules. However, instead of using linear or shallow features on the 3D mesh, our modules take place in latent spaces learned by deep neural networks. This enables capturing of complex non-linear effects, and producing facial animation with a new level of realism.

5.3.2 Deep Codec Avatars

Human perception is particularly sensitive to detecting the realism of facial animation, e.g. the well-known Uncanny Valley Effect [98]. Traditional approaches such as morphable 3D models usually fail to pass the uncanny valley. Recent deep codec avatars have brought new hope to overcome this issue. Lombardi et al. [70] introduced Deep Appearance Models (DAM) that use deep Variational Auto-Encoders (VAE) [54] to jointly encode and decode facial geometry and view-dependent appearances into a latent code. The usage of deep VAE enabled capturing complex non-linear animation effects, while producing a smooth and compact latent representation. View-dependent texture enabled modeling view-dependent effects such as specularity, as well as allowing correcting from imperfect geometry estimation. Their approach enabled realistic facial animation without relying on expensive light-transport rendering. Recently, Lombardi et al. [71] extended their prior work to Neural Volumes (NV). They presented a dynamic, volumetric representation learned through encoder-decoder neural networks using a differentiable ray-marching algorithm. Their approach circumvents the difficulties in conventional mesh-based representation and does not require explicit reconstruction or tracking. The work in [70] is an important cornerstone that our work is built upon. Our work differs in that our main focus is producing accurate facial animation in realtime.

5.3.3 VR Telepresence

The previous work that is the most closely related to ours is Wei et al. [125]. VR telepresence presents many unique challenges due to its novel hardware setup, e.g. unaccommodating camera views that can only see parts of the face, as well as the image domain gap from head-mounted cameras to realistic avatars. In Thies et al. [114] and Cao et al. [19], a well-posed, unobstructed camera is used to provide image input for realtime reenactment. VR telepresence is different in that clear view of the full face is unavailable because the user has to wear the VR headset. Li et al. [61] presented an augmented VR headset that contains an outreaching arm holding an RGB-D camera. Lombardi et al. [70] used cameras mounted on a commodity VR headset, where telepresence was achieved by image-based rendering to create synthetic head-mounted images, and learning a common representation of real and synthetic images. Wei et al. [125] extended this idea by using CycleGAN [136] to further alleviate the image

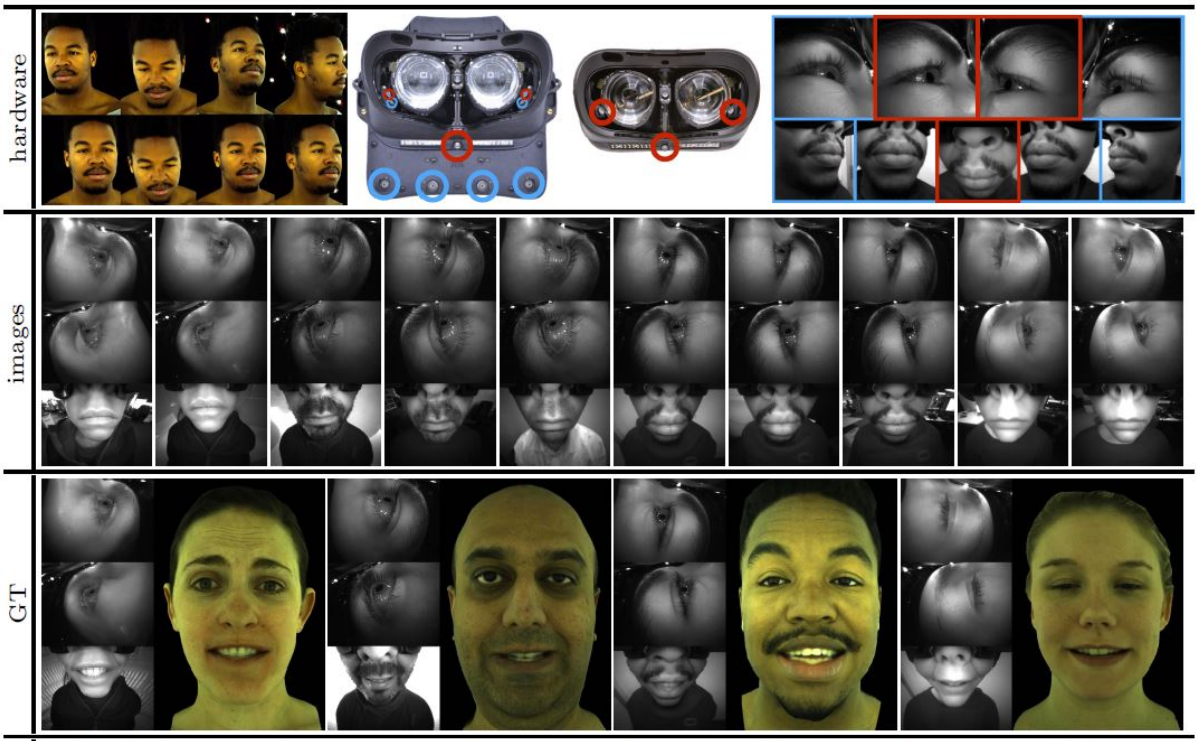


Figure 5.3: Hardware and dataset examples. First row: capture dome [70], training headset, tracking headset, and sample images [125]. Second row: head-mounted camera examples of left eye, right eye, and mouth from different capture runs. Third row: examples of head-mounted images and ground-truth correspondences. Last row: dataset statistics.

domain gap. A training headset with 9 cameras was used to establish correspondence, while a standard headset with 3 cameras was used to track the face. Existing work in VR telepresence are based on holistic face modeling. In our work, we revive the classic module-based approach and combine it with codec avatars to achieve expressive and robust VR telepresence.

5.4 Hardware and Dataset

In this section, we first describe our hardware setup. After that, we provide more details about the dataset that we use to train and evaluate our model.

High quality facial image data is the foundation of realistic facial animation. For this purpose, we use a large multi-camera dome that contains 40 cameras capturing images at a 2560×1920 resolution. Cameras lie on the frontal hemisphere of the face at a distance of about 1 meter. The cameras are synchronized with a frame rate of 30fps. 200 LED lights are directed at the face to create uniform illumination. To collect headset images, we use a two-headset design that has a training headset and a tracking headset. The training headset contains 9 cameras, which ensures establishing high-quality expression between head-mounted cameras and the avatar. The tracking headset contains a subset of 3 cameras, it is a consumer-friendly design with minimally intrusive cameras for real-time telepresence.

The images are captured by IR cameras with a resolution of 640×480 and frame rate of 30fps (down-sampled from 90fps). We refer to [70] and [125] for more details about the camera dome and the VR headsets.

To obtain the high-fidelity facial avatar, we train a person-specific view-dependent VAE using 3D face meshes reconstructed and tracked from dome-captured data similar to [70]. The mesh contains 7306 vertices and a texture map of 1024×1024. We then use the method in [125] to establish correspondences between training headset frames and avatar latent codes. We decode these corresponding latent codes into 3D face meshes. These serve as the ground truth outputs in our dataset. The 9-camera training headset is only used for obtaining accurate ground truth. Both training and evaluation of our model in later sections only uses the subset of 3 cameras on the tracking headset. Note that the method for establishing correspondences provides accurate ground truth output, but it is infeasible to real-time animation due to its expensive computational budget in image domain transfer and iterative image-based synthesis.

We construct a dataset that covers common variations in practical usage scenarios: varying users, varying headsets and varying environments with different backgrounds and lighting conditions. Fig. 5.3 shows some example data and statistics of the dataset. Specifically, our dataset contains four different users and for the headset recording part, we used three headsets and captured in total 14 sessions (half an hour for each session) in three different indoor environments: a small room with weak light, an office environment in front of a desk under bright white light, and in front of a large display screen. In the last environment, we capture some sections while playing random high-frequency flashing patterns on the screen, so that to facilitate the evaluation under extreme lighting condition. Sessions of the same person may be captured on different dates that are months apart, resulting in potential appearance changes in the captured data. For example, for one person, we captured heavy beard in some sessions and the other sessions are captured after he shaved.

For each dome capture or headset capture, we record a predefined set of 73 unique facial expressions, recitation of 50 phonetically-balanced sentences, two range-of-motion sequences where the user is asked to move jaw or whole face randomly with maximum extend, and 5-10 minutes conversation. Finally, we split the dataset by assigning one full headset capture session of each person as the testing set. This means the testing set does not have overlap in terms of capture sessions. We use only sequences of sentence reading and conversation for testing as they reflect the usual facial behaviors of a user in social interactions.

5.5 Method

The goal of our head-only VR telepresence system is to faithfully reconstruct full faces from images captured by the headset-mounted cameras in realtime. In this section, we will first describe the formulation of our model, followed by two important techniques that are important for successfully training the models and lastly implementation details.

5.5.1 Model Formulation

We denote the images captured by the headset-mounted cameras at each time instance as $\mathbf{X}=\{\mathbf{x}_k \mid k \in \{1, ..., K\}\}$, with K the total number of cameras and $\mathbf{x}_k \in \mathbb{R}^I$ where I is the number of pixels in each image. Note the time subscript t is omitted to avoid notation clutter. We denote $\mathbf{v} \in \mathbb{R}^3$ as the direction

from which the avatar is to be viewed in VR. Given $\mathbf{X}$, the telepresence system needs to compute the view-dependent output $\mathbf{y_v}$ which contains the facial geometry $\mathbf{y}^g \in \mathbb{R}^{3G}$ of 3D vertex positions and the facial texture $\mathbf{y}^t_{\mathbf{v}} \in \mathbb{R}^{3T}$ corresponding to view direction $\mathbf{v}$. G and T are the number of vertices on the facial mesh and number of pixels on the texture map respectively.

The holistic Codec Avatar (CA) [70, 125] is formulated as a view-dependent encoder-decoder framework, i.e.

$$\mathbf{y_v} = \mathcal{D}(\mathbf{c}, \mathbf{v}), \ \mathbf{c} = \mathcal{E}(\mathbf{X}) \tag{5.1}$$

where headset cameras' images $\mathbf{X}$ are feed into an image encoder $\mathcal{E}$ to produce the expression code of the whole face, and a decoder $\mathcal{D}$ produces the view-dependent 3D face. $\mathcal{D}$ is trained on dome-captured data to ensure animation quality, and $\mathcal{E}$ is trained using the correspondences between $\mathbf{X}$ and $\mathbf{c}$ that are established using the method in [125]. Because $\mathcal{D}$ is a neural network trained with a limited set of facial expressions, CA has limited expressiveness in realtime animation due to the long tail distribution nature of human facial expressions. In this work we propose Modular Codec Avatar (MCA), where 3D face modules are estimated by an image encoder followed by a synthesizer from each headset camera view, and blended together to form the final face:

$$\mathbf{y_v} = \sum_{k=1}^{K} \mathbf{w}_k \odot \mathcal{D}_k\left(\mathbf{c}_k^{\text{full}}, \mathbf{v}\right) \tag{5.2}$$

$$\left[\mathbf{c}_k^{\text{full}}, \mathbf{w}_k\right] = \mathcal{S}_k\left(\mathbf{c}_k^{\text{part}}\right), \quad \mathbf{c}_k^{\text{part}} = \mathcal{E}_k\left(\mathbf{x}_k\right) \tag{5.3}$$

where each camera view $\mathbf{x}_k$ is processed separately. This computation consists of three steps. Firstly, a modular image encoder $\mathcal{E}_k$ estimates the modular expression code $\mathbf{c}_k^{\text{part}}$ which only models the facial part visible in the kth camera view, e.g., left eye. Subsequently, a modular synthesizer $\mathcal{S}_k$ estimates a latent code for the full-face denoted as $\mathbf{c}_k^{\text{full}}$, based only on the information from $\mathbf{c}_k^{\text{part}}$. The synthesizer also estimates blending weights $\mathbf{w}_k$. Lastly, we aggregate the results from all K modules to form the final face: decode the 3D modular faces using $\mathcal{D}_k$, and blend them together using the adaptive weights. $\odot$ is element-wise multiplication. In this way, MCA learns part-wise expressions inside each face module, while keeping full flexibility of assembling different modules together.

The objective function for train the MCA model consists of three loss terms over final reconstruction and intermediate latent codes:

$$\mathcal{L}_{\text{MCA}} = \|\mathbf{y}_{\mathbf{v}_0} - \hat{\mathbf{y}}_{\mathbf{v}_0}\|_2 + \lambda_1 \sum_{k=1}^{K} \left\|\mathbf{c}_k^{\text{full}} - \hat{\mathbf{c}}\right\|_2 + \lambda_2 \sum_{k=1}^{K} \left\|\mathbf{c}_k^{\text{part}} - \hat{\mathbf{c}}_k^{\text{part}}\right\|_2 \tag{5.4}$$

where $\hat{\mathbf{y}}_{\mathbf{v}_0}$, $\hat{\mathbf{c}}$, and $\hat{\mathbf{c}}_k^{\text{part}}$ denote different supervision signals. The first term measures reconstruction error in facial geometry and texture. We set $\hat{\mathbf{y}}_{\mathbf{v}_0} = \mathcal{D}(\hat{\mathbf{c}}, \mathbf{v}_0)$ as the facial geometry and texture decoded from the supervision code $\hat{\mathbf{c}}$ from frontal view direction $\mathbf{v}_0$. Note the supervision $\hat{\mathbf{c}}$ is estimated from the correspondence stage using the method in [125]. We impose equal weights for reconstruction error in geometry and texture. For the texture, we average pool the reconstruction errors of the pixels corresponding to the same closest mesh vertex instead of averaging over all pixels on the texture map. This vertex-based pooling ensures that texture loss has a geometrically uniform impact. The second term encourages each module to produce independent estimation of the correct full-face and the last term directs each encoder to produce correct modular expression code. Section 5.5.2 describes in detail

how we generate the supervision $\hat{\mathbf{c}}_k^{\text{part}}$.

5.5.2 Exemplar-based Latent Alignment

The two latent codes $\mathbf{c}_k^{\text{part}}$ and $\mathbf{c}_k^{\text{full}}$ have different purposes. $\mathbf{c}_k^{\text{part}}$ represents information only within its responsible modular region, while $\mathbf{c}_k^{\text{full}}$ further synthesizes a full-face based on the single-module expression information. It is important to have $\mathbf{c}_k^{\text{part}}$, because otherwise the modules will only collectively try to recover the same full-face code through $\mathbf{c}_k^{\text{full}}$, which essentially degrades to the holistic CA. The key to ensure the effectiveness of $\mathbf{c}_k^{\text{part}}$ is through crafting proper supervision signal $\hat{\mathbf{c}}_k^{\text{part}}$. The main challenge is $\hat{\mathbf{c}}_k^{\text{part}}$ resides in an inexplicitly defined latent space. We address this problem with exemplar-based latent alignment.

To obtain $\hat{\mathbf{c}}_k^{\text{part}}$, we train a separate VAE for each module from dome-captured data. It has a similar architecture as CA, but only uses the region within the module. This is realized by applying a modular mask on both the input and output of the VAE. We denote the masked modular VAE decoder as $\mathcal{D}_k^{\text{mask}}$, and the set of codes corresponding to dome-captured modular faces as $\mathbf{C}_k^{\text{mask}}$. Next, we need to obtain $\hat{\mathbf{c}}_k^{\text{part}}$ for head-images to train MCA. A seemingly effective approach is directly using the VAE encoder, or minimizing output residual error with ground-truth using the decoder. However, because the VAE is over-parameterized and the codec space has redundancy, we found these approach produces codes that are distributed arbitrarily in the latent space, failing to carry effective modular information. We use exemplar-based latent alignment to overcome this issue. It avoids the noise in code, and calibrates part codes of head-mounted data from different capture runs to take similar values as the fixed set of dome-captured code, i.e.

$$\hat{\mathbf{c}}_k^{\text{part}} = \underset{\mathbf{c}\in\mathbf{C}_k^{\text{mask}}}{\operatorname{argmin}} \left\| \mathcal{D}_k^{\text{mask}}(\mathbf{c}, \mathbf{v}_0) - \hat{\mathbf{y}}_{\mathbf{v}_0,k}^{\text{mask}} \right\|_2 \tag{5.5}$$

where $\hat{\mathbf{y}}_{\mathbf{v}_0,k}^{\text{mask}}$ is the modular masked $\hat{\mathbf{y}}_{\mathbf{v}_0}$. This differs from pure image-based synthesis in that result must come from a set of known dome-captured exemplars $\mathbf{C}_k^{\text{mask}}$. The resulting $\hat{\mathbf{c}}_k^{\text{part}}$ is then used in Eq.(4) to train MCA.

5.5.3 Modulated Adaptive Blending

The blending weights $\mathbf{w}_k$ can be fully learned from data. However, we find automatically learned blending weights lead to animation artifacts. This is because module correlation exists in training data, e.g. left and right eyes often open and close together. Therefore, the dominant module interchanges between left and right eyes across nearby pixels, which results in jigsaw-like artifacts in the eyeball region. To overcome this issue and promote spatial coherence in the modular blending weights, we inject multiplicative and additive modulation signals to the adaptively learned blending weights. This ensures blending weight is constant near the module centroid, and the importance of adaptively learned weights gradually increases, i.e.

$$\mathbf{w}_k = \frac{1}{\mathbf{w}} \odot \left(\mathbf{w}_k^{\mathcal{S}} \odot e^{-\frac{\max\{\|\mathbf{u}-\overline{\mathbf{u}}_k\|^2 - a_k, 0\}}{\sigma^2}} + b_k \mathbb{1}\left\{ \|\mathbf{u}-\overline{\mathbf{u}}_k\|^2 \le a_k \right\} \right) \tag{5.6}$$

where $\mathbf{w}_k^{\mathcal{S}}$ denotes the adaptive blending weights produced by the synthesizer $\mathcal{S}_k$, $\mathbf{u}$ denotes the 2D texture map coordinates corresponding to each vertex, $\overline{\mathbf{u}}_k$, a_k, and b_k are constants denoting the module's centroid, area in the texture map, and constant amplitude within its area. $\mathbf{w}$ is computed vertex-wise

to normalize the blending weights across all modules.

5.5.4 Implementation Details

We use K=3 modules following our hardware design, with three head-mounted cameras capturing left eye, right eye, and mouth. Each $\mathbf{x}_k$ is resized to a dimension of 256×256. We use a latent dimension of 256 for both $\hat{\mathbf{c}}_k^{\text{part}}$ and $\hat{\mathbf{c}}$. Similar neural network architectures to [125] are used for our encoder $\mathcal{E}_k$ and decoder $\mathcal{D}_k$. For the decoder, we reduce the feature dimensions to remedy computation cost due to decoding K times. We also set all $\mathcal{D}_k$ to share the same weights to save the capacity requirement for storage and transmission of the model. The synthesizer $\mathcal{S}_k$ consists of three temporal convolution layers [9] (TCN), with connections to a small temporal receptive field of 4 previous frames. The blending weights $\mathbf{w}_k^{\mathcal{S}}$ is predicted through transposed convolution layers with a sigmoid function at the end to fit the texture map resolution. The texture weights were then reordered to produce geometry weights using the correspondence between vertices and texture map coordinates. We also add cross-module skip connections that concatenates the last layer features in $\mathcal{E}_k$ to allow the model to exploit correlations between images from different headset-mounted cameras.

We train the model with the Adam optimizer with both λ_1 and λ_2 set to 1. We augment the headset images by randomly crop, zoom, and rotate the images. We also add random gaussian noises to the modular code $\hat{\mathbf{c}}_k^{\text{part}}$ with diagonal covariance determined by distance to the closest neighbour to prevent overfitting. The training is completed using 4 Tesla V100 GPUs. In run time, the telepresence system using MCA takes in average 42.9ms to produce one frame in VR, achieving photo-realistic facial animation at 30 fps.

5.6 Experiments

In this section, we first report an analytical experiment that illustrates advantage of modular modeling of facial expressions over holistic modeling, which motivates our work. We then provide detailed evaluation results of the full VR telepresence via MCA comparing to via CA. Our experiment evaluates the performance from extensive perspectives, including expression accuracy, improvement consistency and perceptive quality. We also provide detailed ablation study and discuss failure cases. Finally, we show two extensive applications of our method which may be useful under certain usage scenarios.

5.6.1 Facial Expression Modeling: Modular vs. Holistic

In the first experiment, we show the advantage of modeling modular expressions in real world VR telepresence scenarios. Towards this goal, we train a holistic VAE [70, 125] as well as modular VAEs on dome-captured data. For both approach, we apply agglomerative clustering on the resulting latent codes of the training set data with varying number of clusters to represent the corresponding model's capacity of representing facial expressions. Recall that for headset-captured data, we have their estimated *ground truth* 3D face through the correspondence stage using the method in [125]. We then use these 3D faces to retrieve the closest cluster centers by matching the pixel values within each modular region of the rendered frontal faces, and report the root mean squared error (RMSE) of the pixel values in Fig. 5.4. For fair comparison, we use K=3 times more clusters for the CA baseline. Fig. 5.4 shows the result.

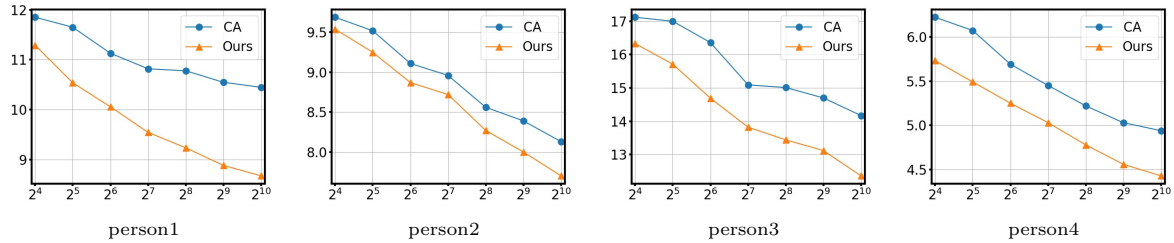


Figure 5.4: Comparing holistic versus modular by isolating and evaluating the importance expressiveness. X-axis shows different capacities of the face expressions by the number of clusters. Y-axis shows the RMSE photometric error. Note that CA uses proportionally K times more clusters than each module of MCA for fair comparison.

It can be seen from Fig. 5.4 that MCA consistently produces lower matching errors than CA throughout all persons and model capacity levels. Intuitively this means the users show many expressions when wearing the headset that are holistically unseen in the dome-captured data, but seen in the modular sense. The gap between CA and MCA increases as the number of clusters increases, indicating MCA's advantage increases as the face modeling becomes more fine-grained. It is then clear that by modeling modular expressions in the MCA, we can more truthfully interpolate/extrapolate facial expressions that are not contained in the dome-captured data, achieving better expressiveness in telepresence.

5.6.2 Full VR Telepresence

We evaluate the performance of the full VR telepresence system. We feed headset images as input to the model, and evaluate the 3D face output against the ground-truth. Six metrics are used for comprehensive evaluation and the results are reported in Table 5.1.

Expression Accuracy

The most important metrics are MAE (i.e. Mean Absolute Error) and RMSE on pixels in the rendered frontal view face, as they directly and metrically measure the accuracy of VR telepresence. To further decompose the error into geometry and texture components, we compute RMSE on vertex position (Geo.) and texture map (Tex.) respectively as well. MCA *consistently outperform* CA on all these metrics on almost all identities' test data. These results suggest that MCA more truthfully recover the 3D face given only the partial views of the face from headset-camera images than CA, leading to more expressive and robust telepresense.

Improvement Consistency

As these metric values are computed from average of all frames, we need to verify that the improvement of MCA is not due to large performance differences on only a small set of frames. To do that, we compute the percentage of frames for which MCA produces more accurate results than CA and vice versa. This is reported as %-better in Table 5.1. From the results, it is clear that MCA *consistently outperform* CA across frames. For example, for person3, on 99.7% frames MCA produces more accurate results than CA.

	MAE↓		RMSE↓		Geo.↓		Tex.↓		%-better↑		SSIM↑	
method	CA	Ours	CA	Ours	CA	Ours	CA	Ours	CA	Ours	CA	Ours
person1	8.82	**8.69**	7.67	**7.47**	1.26	**1.14**	3.40	**3.02**	36.3	**63.7**	0.954	**0.957**
person2	4.44	**4.26**	4.00	**3.84**	1.82	**1.46**	2.05	**2.04**	27.3	**72.7**	0.949	**0.951**
person3	9.09	**6.97**	8.36	**6.66**	1.14	**0.84**	4.58	**3.43**	0.3	**99.7**	0.933	**0.942**
person4	3.33	**3.21**	3.08	**3.04**	**0.54**	0.64	0.86	**0.85**	41.1	**58.9**	0.984	0.984
overall	6.54	**6.17**	5.81	**5.48**	1.37	**1.17**	2.72	**2.44**	29.3	**70.7**	0.953	**0.956**

Table 5.1: Quantitative results of our main experiment evaluating the full VR telepresence system robustness. MCA outperforms CA across various metrics. Please refer to the text for details.

Perceptive Quality

We also want to verify that such accuracy improvements align with human perception, i.e. whether a user may actually feel the improvement. Quantitatively, we compute structural similarity index on the grey-level frontal rendering (SSIM) [124] which is a perception-based metric that considers image degradation as perceived change in structural information, while also incorporating important perceptual phenomena. These results are reported in the last column in Table 5.1: MCA outperforms CA on three persons while is on-par on the last one. This hints that the accuracy improvements of MCA over CA align with perception improvement. Fig. 5.5 shows a qualitative comparison between MCA and CA. It can be seen that MCA is better at handling subtle combined expressions, e.g. mouth open while eyes half-open, showing teeth while eyes shut, and mouth stretched while looking down. Renderings from different viewpoint by varying $\mathbf{v}$ are shown in Fig. 5.6. It can be seen that MCA produces consistent results from different viewing directions.

Albation Study

Table 5.2 shows an ablation study of MCA. Ablation factors range from the usage of synthesized blending weights instead of equal weights (blend), training the network end-to-end (end2end), using soft latent part vectors with gaussian noise instead of hard categorical classes (soft-ex.), expanding the dimension of latent codes (dimen.), using skip-module connections so all images are visible to the modular encoder (skip-mod.), and using temporal convolution on the synthesizer (tconv.). It can be seen that blending weights, end-to-end training, and soft latent vectors contribute significantly, while extra connections such as skip and temporal lead to further improvements.

Failure Cases

Fig. 5.7 shows example failure cases. Strong background flash is a challenging problem even for MCA, leading to inaccurate output (left of Fig. 5.7). Although MCA can produce more expressive 3D face, extreme asymmetric expressions like one pupil in the center while the other roll all the way to the corner as in the right of Fig. 5.7 still remains challenging to be faithfully reconstructed.

5.6.3 Extensive Applications

Flexible Animation

Making funny expressions is part of social interaction. The MCA model can naturally better facilitate this task, since it has stronger expressiveness. To showcase this, we shuffle the head-mounted image sequences separately for each module, and randomly match them to simulate flexible expressions. It

blend	end2end	soft-ex.	dimen.	skip-mod.	tconv.	RMSE	$\Delta_{\downarrow}$
-	-	-	-	-	-	7.33	-
✓	-	-	-	-	-	6.67	0.66
✓	✓	-	-	-	-	6.10	0.57
✓	✓	✓	-	-	-	5.71	0.39
✓	✓	✓	✓	-	-	5.64	0.07
✓	✓	✓	✓	✓	-	5.52	0.08
✓	✓	✓	✓	✓	✓	5.48	0.04

Table 5.2: An ablation study of our method showing the progression and contribution to the overall performance improvement on the main RMSE metric.

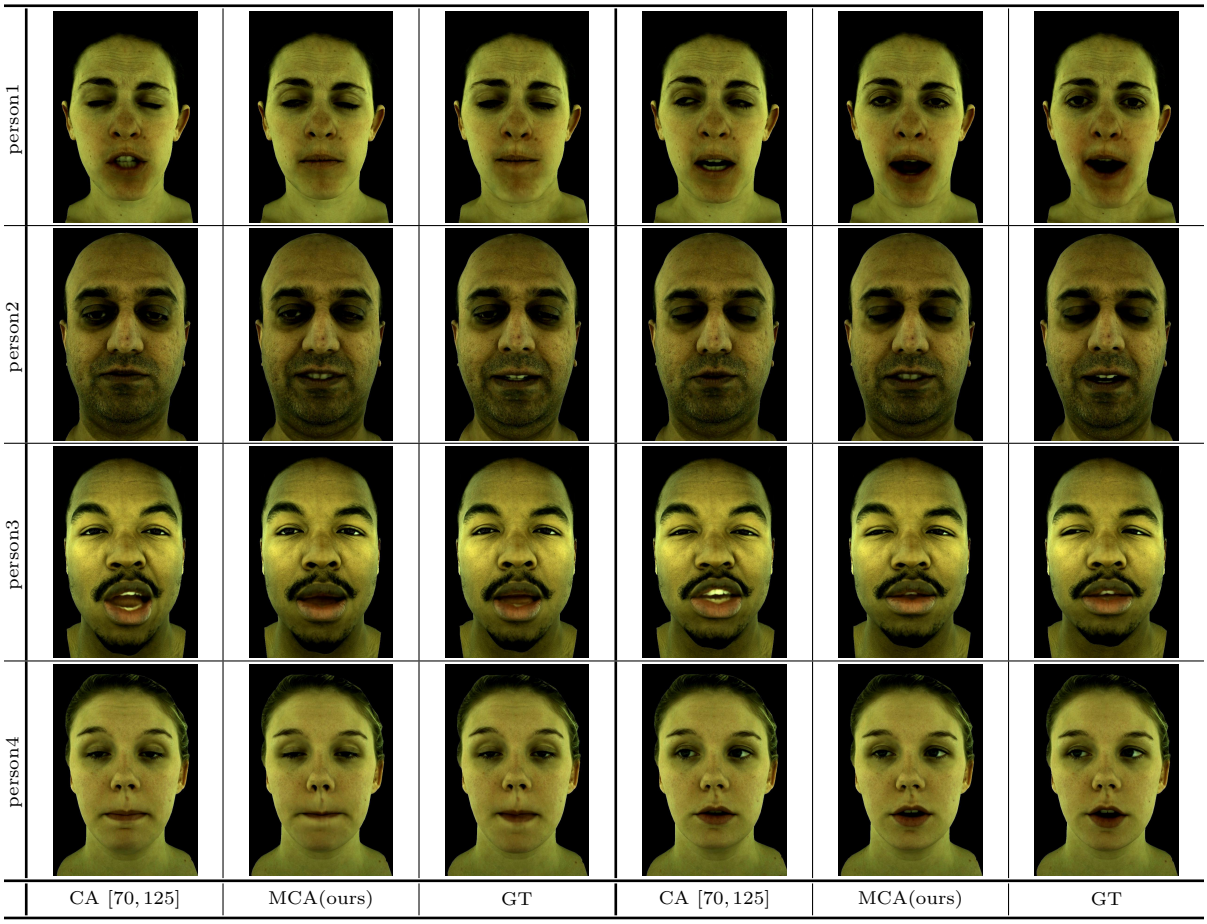


Figure 5.5: Qualitative VR telepresence results. Compared to the holistic approach, MCA handles unseen subtle expressions better due to stronger expressiveness.

can be seen from Fig. 5.8 that MCA produces natural flexible expressions, even though such expressions have never been seen holistically in the training set.

Eye Amplification

In practical VR telepresence, we observe users often do not open their eyes to the full natural extend. This maybe due to muscle pressure from the headset wearing, and display light sources near the eyes. We introduce an eye amplification control knob to address this issue. In MCA, this can be simply accomplished by identifying the base $\mathbf{c}_k^{\text{part}}$ that correspond to closed eye, and amplifying the latent

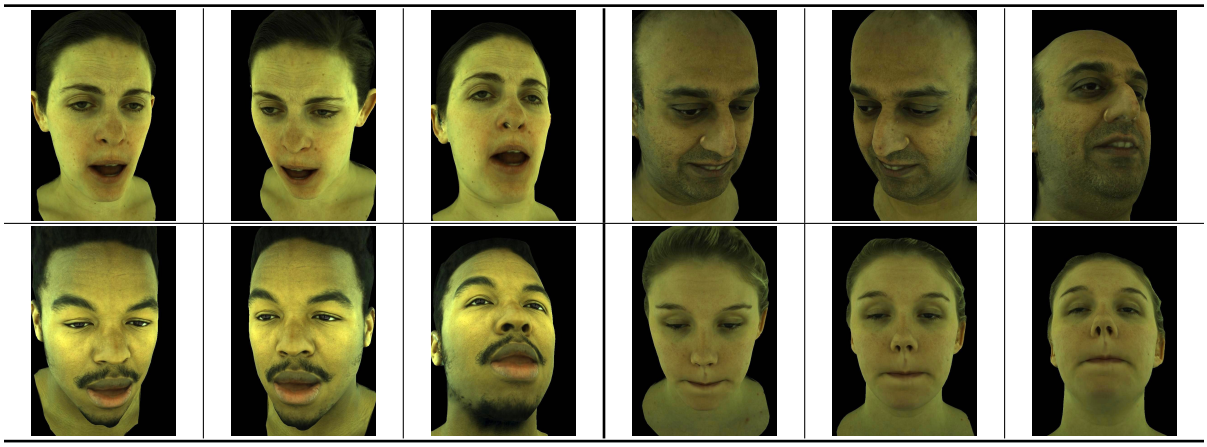


Figure 5.6: Qualitative results of MCA from different viewing directions by varying $\mathbf{v}$. MCA produces natural expressions that are consistent across viewpoints.

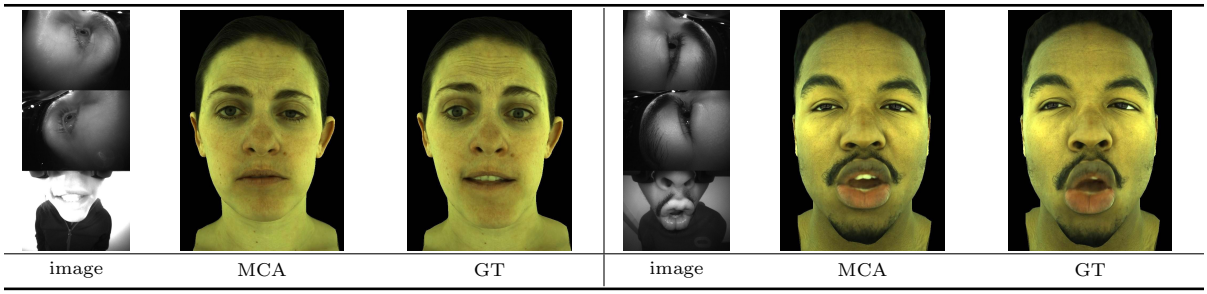


Figure 5.7: Failure cases of MCA. Typical failure cases include interference from strong background flash, extreme asymmetry between the eyes, and weakened motion.

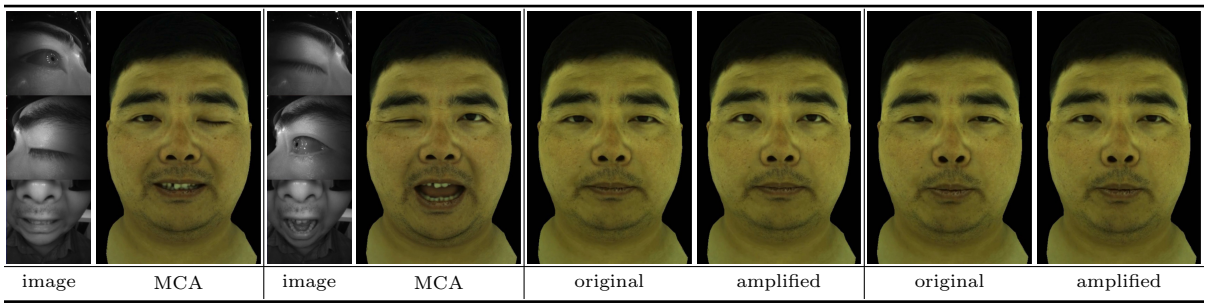


Figure 5.8: Two new applications enabled by the MCA model. Left side shows two examples of flexible animation. Right side shows two examples of eye amplification.

space distance by multiplying a user-provided amplification magnitude. Fig. 5.8 shows examples of amplifying by a factor of 2.

5.7 Conclusion

We addressed the problem of VR telepresence, which aimed to provide remote and immersive face-to-face telecommunication through VR headsets. Codec Avatar (CA) utilized view-dependent neural

networks to achieve realistic facial animation. We presented a new formulation of codec avatar named Modular Codec Avatar (MCA). We were the first to combine classic module-based face modeling with codec avatars in VR telepresence. We presented several important techniques to realize MCA effectively. We demonstrated that MCA achieves improved expressiveness and robustness through experiments on a comprehensive real-world dataset that emulated practical scenarios. New applications in VR telepresence enabled by the proposed model were finally showcased.

Bibliography

[1] Esri: Cityengine software.

[2] The spacenet dataset on aws.

[3] Vicon: The cara system.

[4] David Acuna, Amlan Kar, and Sanja Fidler. Devil is in the edges: Learning semantic boundaries from noisy annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11075–11083, 2019.

[5] David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 859–868, 2018.

[6] Sawsan AlHalawani, Yong-Liang Yang, Peter Wonka, and Niloy J Mitra. What makes london work like london? In *Computer Graphics Forum*, volume 33, pages 157–165. Wiley Online Library, 2014.

[7] Daniel G Aliaga, Carlos A Vanegas, and Bedrich Benes. Interactive example-based urban layout synthesis. *ACM transactions on graphics (TOG)*, 27(5):1–10, 2008.

[8] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[9] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[10] Tadas Baltrušaitis, Peter Robinson, and Louis-Philippe Morency. Openface: an open source facial behavior analysis toolkit. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–10. IEEE, 2016.

[11] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4720–4728, 2018.

[12] Jan Beneš, Alexander Wilkie, and Jaroslav Křivánek. Procedural modelling of urban road networks. In *Computer Graphics Forum*, volume 33, pages 132–142. Wiley Online Library, 2014.

[13] Joan-Isaac Biel and Daniel Gatica-Perez. The youtube lens: Crowdsourced personality impressions and audiovisual analysis of vlogs. *IEEE Transactions on Multimedia*, 15(1):41–55, 2012.

[14] Michael J Black and Yaser Yacoob. Recognizing facial expressions in image sequences using local parameterized models of image motion. *International Journal of Computer Vision*, 25(1):23–48, 1997.

[15] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194, 1999.

[16] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. *arXiv preprint arXiv:1803.00816*, 2018.

[17] Ricardo Cabral and Yasutaka Furukawa. Piecewise planar and compact floorplan reconstruction from images. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 628–635. IEEE, 2014.

[18] Chen Cao, Menglei Chai, Oliver Woodford, and Linjie Luo. Stabilized real-time face tracking via a learned dynamic rigidity prior. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018.

[19] Chen Cao, Qiming Hou, and Kun Zhou. Displaced dynamic expression regression for real-time facial tracking and animation. *ACM Transactions on graphics (TOG)*, 33(4):1–10, 2014.

[20] Chen Cao, Yanlin Weng, Shun Zhou, Yiying Tong, and Kun Zhou. Facewarehouse: A 3d facial expression database for visual computing. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):413–425, 2013.

[21] Lluis Castrejon, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Annotating object instances with a polygon-rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5230–5238, 2017.

[22] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller, and Eugene Zhang. Interactive procedural street modeling. In *ACM SIGGRAPH 2008 papers*, pages 1–10. 2008.

[23] Dominic Cheng, Renjie Liao, Sanja Fidler, and Raquel Urtasun. Darnet: Deep active ray network for building segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7431–7439, 2019.

[24] Hang Chu, Dong Ki Kim, and Tsuhan Chen. You are here: Mimicking the human thinking process in reading floor-plans. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2210–2218, 2015.

[25] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[26] Andrea Cohen, Alexander G Schwing, and Marc Pollefeys. Efficient structured parsing of facades using dynamic programming. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3206–3213, 2014.

[27] Kenneth Mark Colby. Modeling a paranoid mind. *Behavioral and Brain Sciences*, 4(4):515–534, 1981.

[28] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence*, 23(6):681–685, 2001.

[29] Bo Dai, Sanja Fidler, Raquel Urtasun, and Dahua Lin. Towards diverse and natural image descriptions via a conditional gan. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2970–2979, 2017.

[30] Paul E Debevec, Camillo J Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20, 1996.

[31] Ilke Demir, Krzysztof Koperski, David Lindenbaum, Guan Pang, Jing Huang, Saikat Basu, Forest Hughes, Devis Tuia, and Ramesh Raska. Deepglobe 2018: A challenge to parse the earth through satellite images. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 172–17209. IEEE, 2018.

[32] Anthony R Dick, Philip HS Torr, and Roberto Cipolla. Modelling and interpretation of architecture from several images. *International Journal of Computer Vision*, 60(2):111–134, 2004.

[33] Piotr Dollár and C Lawrence Zitnick. Structured forests for fast edge detection. In *Proceedings of the IEEE international conference on computer vision*, pages 1841–1848, 2013.

[34] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

[35] Paul Ekman, Wallace V Freisen, and Sonia Ancoli. Facial signs of emotional experience. *Journal of personality and social psychology*, 39(6):1125, 1980.

[36] Mohamed Elgharib, Mallikarjun BR, Ayush Tewari, Hyeongwoo Kim, Wentao Liu, Hans-Peter Seidel, and Christian Theobalt. Egoface: Egocentric face performance capture and videorealistic reenactment. *arXiv preprint arXiv:1905.10822*, 2019.

[37] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. Worldbrush: Interactive example-based synthesis of procedural virtual worlds. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015.

[38] Tian Feng, Lap-Fai Yu, Sai-Kit Yeung, KangKang Yin, and Kun Zhou. Crowd-driven mid-scale layout design. *ACM Trans. Graph.*, 35(4):132–1, 2016.

[39] Yasutaka Furukawa, Brian Curless, Steven M Seitz, and Richard Szeliski. Reconstructing building interiors from images. In *2009 IEEE 12th International Conference on Computer Vision*, pages 80–87. IEEE, 2009.

[40] Eric Galin, Adrien Peytavie, Eric Guérin, and Bedřich Beneš. Authoring hierarchical road networks. In *Computer Graphics Forum*, volume 30, pages 2021–2030. Wiley Online Library, 2011.

[41] Donya Ghafourzadeh, Cyrus Rahgoshay, Sahel Fallahdoust, Adeline Aubame, Andre Beauchamp, Tiberiu Popa, and Eric Paquette. Part-based 3d face morphable model with anthropometric local control. In *Annual Conference of the European Association for Computer Graphics*, 2020.

[42] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[43] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[45] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

[46] David L Heymann and Nahoko Shindo. Covid-19: what is next for public health? *The Lancet*, 395(10224):542–545, 2020.

[47] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[48] Namdar Homayounfar, Wei-Chiu Ma, Shrinidhi Kowshika Lakshmikanth, and Raquel Urtasun. Hierarchical recurrent attention networks for structured online maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3417–3426, 2018.

[49] Liwen Hu, Shunsuke Saito, Lingyu Wei, Koki Nagano, Jaewoo Seo, Jens Fursund, Iman Sadeghi, Carrie Sun, Yen-Chun Chen, and Hao Li. Avatar digitization from a single image for real-time rendering. *ACM Transactions on Graphics (TOG)*, 36(6):1–14, 2017.

[50] Satoshi Ikehata, Hang Yang, and Yasutaka Furukawa. Structured indoor modeling. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1323–1331, 2015.

[51] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[52] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[53] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[54] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[55] Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[57] Abhijit Kundu, Yin Li, Frank Dellaert, Fuxin Li, and James M Rehg. Joint semantic segmentation and 3d reconstruction from monocular video. In *European Conference on Computer Vision*, pages 703–718. Springer, 2014.

[58] Lubor Ladicky, Jianbo Shi, and Marc Pollefeys. Pulling things out of perspective. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 89–96, 2014.

[59] Sergey Levine, Philipp Krähenbühl, Sebastian Thrun, and Vladlen Koltun. Gesture controllers. In *ACM SIGGRAPH 2010 papers*, pages 1–11. 2010.

[60] Sergey Levine, Christian Theobalt, and Vladlen Koltun. Real-time prosody-driven synthesis of body language. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–10. 2009.

[61] Hao Li, Laura Trutoiu, Kyle Olszewski, Lingyu Wei, Tristan Trutna, Pei-Lun Hsieh, Aaron Nicholls, and Chongyang Ma. Facial performance sensing head-mounted display. *ACM Transactions on Graphics (ToG)*, 34(4):1–9, 2015.

[62] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*, 2015.

[63] Jiwei Li, Michel Galley, Chris Brockett, Georgios P Spithourakis, Jianfeng Gao, and Bill Dolan. A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*, 2016.

[64] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[65] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[66] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. Polymapper: Extracting city maps using polygons. *arXiv preprint arXiv:1812.01497*, 2, 2018.

[67] Huan Ling, Jun Gao, Amlan Kar, Wenzheng Chen, and Sanja Fidler. Fast interactive object annotation with curve-gcn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5257–5266, 2019.

[68] Beyang Liu, Stephen Gould, and Daphne Koller. Single image depth estimation from predicted semantic labels. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1253–1260. IEEE, 2010.

[69] Chenxi Liu, Alexander G Schwing, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Rent3d: Floor-plan priors for monocular layout estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3413–3421, 2015.

[70] Stephen Lombardi, Jason Saragih, Tomas Simon, and Yaser Sheikh. Deep appearance models for face rendering. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.

[71] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Transactions on Graphics (TOG)*, 38(4):65, 2019.

[72] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[73] Diego Marcos, Devis Tuia, Benjamin Kellenberger, Lisa Zhang, Min Bai, Renjie Liao, and Raquel Urtasun. Learning deep structured active contours end-to-end. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8877–8885, 2018.

[74] Anjelo Martinović, Markus Mathias, Julien Weissenberg, and Luc Van Gool. A three-layered approach to facade parsing. In *European conference on computer vision*, pages 416–429. Springer, 2012.

[75] Gellért Máttyus, Wenjie Luo, and Raquel Urtasun. Deeproadmapper: Extracting road topology from aerial images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3438–3446, 2017.

[76] Branislav Mičušík and Jana Košecká. Multi-view superpixel stereo in urban environments. *International journal of computer vision*, 89(1):106–119, 2010.

[77] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. In *ACM Transactions on Graphics (TOG)*, volume 26, page 85. ACM, 2007.

[78] Przemyslaw Musialski, Peter Wonka, Daniel G Aliaga, Michael Wimmer, Luc Van Gool, and Werner Purgathofer. A survey of urban reconstruction. In *Computer graphics forum*, volume 32, pages 146–177. Wiley Online Library, 2013.

[79] Koki Nagano, Jaewoo Seo, Jun Xing, Lingyu Wei, Zimo Li, Shunsuke Saito, Aviral Agarwal, Jens Fursund, Hao Li, Richard Roberts, et al. pagan: real-time avatars using dynamic textures. *ACM Trans. Graph.*, 37(6):258–1, 2018.

[80] Thomas Neumann, Kiran Varanasi, Stephan Wenger, Markus Wacker, Marcus Magnor, and Christian Theobalt. Sparse localized deformation components. *ACM Transactions on Graphics (TOG)*, 32(6):1–10, 2013.

[81] Gen Nishida, Ignacio Garcia-Dorado, and Daniel G Aliaga. Example-driven procedural urban roads. In *Computer Graphics Forum*, volume 35, pages 5–17. Wiley Online Library, 2016.

[82] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L Davidson, Sameh Khamis, Mingsong Dou, et al. Holoportation: Virtual 3d teleportation in real-time. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 741–754, 2016.

[83] Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, 2001.

[84] Chi-Han Peng, Yong-Liang Yang, Fan Bao, Daniel Fink, Dong-Ming Yan, Peter Wonka, and Niloy J Mitra. Computational network design from functional specifications. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[85] Timo Pylvänäinen, Jérôme Berclaz, Thommen Korah, Varsha Hedau, Mridul Aanjaneya, and Radek Grzeszczuk. 3d city modeling from street-level data for augmented reality applications. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 238–245. IEEE, 2012.

[86] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256, 1972.

[87] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

[88] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[89] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7008–7024, 2017.

[90] Alan Ritter, Colin Cherry, and William B Dolan. Data-driven response generation in social media. In *Proceedings of the conference on empirical methods in natural language processing*, pages 583–593. Association for Computational Linguistics, 2011.

[91] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[92] Nikolay Savinov, Lubor Ladicky, Christian Hane, and Marc Pollefeys. Discrete optimization of ray potentials for semantic 3d reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5511–5518, 2015.

[93] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.

[94] Alexander G Schwing, Sanja Fidler, Marc Pollefeys, and Raquel Urtasun. Box in the box: Joint 3d layout and object reasoning from single images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 353–360, 2013.

[95] Alexander G Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Efficient structured prediction for 3d indoor scene understanding. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2815–2822. IEEE, 2012.

[96] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[97] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[98] Jun'ichiro Seyama and Ruth S Nagayama. The uncanny valley: Effect of realism on the impression of artificial human faces. *Presence: Teleoperators and virtual environments*, 16(4):337–351, 2007.

[99] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1145–1153, 2017.

[100] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pages 412–422. Springer, 2018.

[101] Sudipta Sinha, Drew Steedly, and Rick Szeliski. Piecewise planar stereo for image-based rendering. 2009.

[102] Sudipta N Sinha, Drew Steedly, Richard Szeliski, Maneesh Agrawala, and Marc Pollefeys. Interactive 3d architectural modeling from unordered photo collections. *ACM Transactions on Graphics (TOG)*, 27(5):1–10, 2008.

[103] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[104] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[105] Supasorn Suwajanakorn, Steven M Seitz, and Ira Kemelmacher-Shlizerman. What makes tom hanks look like tom hanks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3952–3960, 2015.

[106] Supasorn Suwajanakorn, Steven M Seitz, and Ira Kemelmacher-Shlizerman. Synthesizing obama: learning lip sync from audio. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

[107] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[108] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[109] Makarand Tapaswi, Yukun Zhu, Rainer Stiefelhagen, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Movieqa: Understanding stories in movies through question-answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4631–4640, 2016.

[110] Olivier Teboul, Loic Simon, Panagiotis Koutsourakis, and Nikos Paragios. Segmentation of building facades using procedural shape priors. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3105–3112. IEEE, 2010.

[111] Seth J Teller and Carlo H Séquin. Visibility preprocessing for interactive walkthroughs. *ACM SIGGRAPH Computer Graphics*, 25(4):61–70, 1991.

[112] J Rafael Tena, Fernando De la Torre, and Iain Matthews. Interactive region-based linear 3d face models. In *ACM SIGGRAPH 2011 papers*, pages 1–10. 2011.

[113] Ayush Tewari, Florian Bernard, Pablo Garrido, Gaurav Bharaj, Mohamed Elgharib, Hans-Peter Seidel, Patrick Pérez, Michael Zollhofer, and Christian Theobalt. Fml: Face model learning from videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10812–10822, 2019.

[114] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2387–2395, 2016.

[115] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 6(Sep):1453–1484, 2005.

[116] Carlos A Vanegas, Daniel G Aliaga, Bedrich Benes, and Paul A Waddell. Interactive design of urban spaces using geometrical and behavioral modeling. *ACM transactions on graphics (TOG)*, 28(5):1–10, 2009.

[117] Vivek Verma, Rakesh Kumar, and Stephen Hsu. 3d building detection and modeling from aerial lidar data. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2213–2220. IEEE, 2006.

[118] Paul Vicol, Makarand Tapaswi, Lluis Castrejon, and Sanja Fidler. Moviegraphs: Towards understanding human-centric situations from videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8581–8590, 2018.

[119] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[120] Lu Wang and Ulrich Neumann. A robust approach for automatic registration of aerial images with untextured aerial lidar data. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2623–2630. IEEE, 2009.

[121] Shenlong Wang, Min Bai, Gellert Mattyus, Hang Chu, Wenjie Luo, Bin Yang, Justin Liang, Joel Cheverie, Sanja Fidler, and Raquel Urtasun. Torontocity: Seeing the world with a million eyes. *arXiv preprint arXiv:1612.00423*, 2016.

[122] Shenlong Wang, Sanja Fidler, and Raquel Urtasun. Holistic 3d scene understanding from a single geo-tagged image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3964–3972, 2015.

[123] Shenlong Wang, Sanja Fidler, and Raquel Urtasun. Lost shopping! monocular localization in large indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2695–2703, 2015.

[124] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[125] Shih-En Wei, Jason Saragih, Tomas Simon, Adam W Harley, Stephen Lombardi, Michal Perdoch, Alexander Hypes, Dawei Wang, Hernan Badino, and Yaser Sheikh. Vr facial animation via multiview image translation. *ACM Transactions on Graphics (TOG)*, 38(4):1–16, 2019.

[126] Joseph Weizenbaum. Elizaa computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[127] Tomás Werner and Andrew Zisserman. New techniques for automated architectural reconstruction from photographs. In *European conference on computer vision*, pages 541–555. Springer, 2002.

[128] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[129] Jianxiong Xiao, Tian Fang, Peng Zhao, Maxime Lhuillier, and Long Quan. Image-based street-side city modeling. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–12. 2009.

[130] Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. Urban pattern: Layout design by hierarchical domain splitting. *ACM Transactions on Graphics (TOG)*, 32(6):1–12, 2013.

[131] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

[132] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2403–2412, 2018.

[133] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[134] Lukas Zebedin, Joachim Bauer, Konrad Karner, and Horst Bischof. Fusion of feature-and area-based information for urban buildings modeling from aerial imagery. In *European conference on computer vision*, pages 873–886. Springer, 2008.

[135] Jianming Zhang, Stan Sclaroff, Zhe Lin, Xiaohui Shen, Brian Price, and Radomir Mech. Minimum barrier salient object detection at 80 fps. In *Proceedings of the IEEE international conference on computer vision*, pages 1404–1412, 2015.

[136] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[137] Xiangxin Zhu and Deva Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *2012 IEEE conference on computer vision and pattern recognition*, pages 2879–2886. IEEE, 2012.

[138] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

[139] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *European conference on computer vision*, pages 391–405. Springer, 2014.