2016年度 学士論文

Sketch-Based 3D Modeling via Deep Convolutional Neural Networks

守矢拓海

慶應義塾大学環境情報学部

Abstract

昨今の 3D プリンタの普及によって, いままでものづくりに深く関わったことがない人も 3D プリンタを使う機会が増えている．しかし, ユーザが欲しい 3D モデルを出力するためには, まず CAD を使ってモデリングを行い, データを作成しなければならない．いままで CAD を使ったことがない人にとってはこれが 3D プリントするまでの大きな壁となっている．そこで, 本研究ではスケッチによる 3D モデリングシステムを開発する．このシステムは, ユーザがインターフェース上で絵を描くと, 入力された絵をもとに 3D モデルを生成する．具体的には, Generative Adversarial Networks を用いてスケッチからボクセルモデルの生成を行う．従来の Sketch-Based Modeling の研究ではアルゴリズムの性質上, 1 枚のスケッチに対して 1 個の 3D モデルが作られるようになっていたが, 本研究で提案する手法を用いれば 1 枚のスケッチに対して複数個の 3D モデルを生成できるようになる．実験では, どこまで人間の作った 3D モデルと判別のつかないものを生成できるか評価するために, 生成した 3D モデルと人間が作った 3D モデルを並べ, どれが人間が作ったものかを実験参加者に当ててもらうテストを行った.

Keywords: Digital fabrication, 3D modeling, sketching, generative models.

本論文の最新バージョンは http://fab.sfc.keio.ac.jp で閲覧可能.

Contents

1 Introduction

昨今の 3D プリンタの普及によって, いままでものづくりに深く関わったことがない人も 3D プリンタを使う機会が増えている. 例えば, 慶應大学湘南藤沢キャンパスのメディアセンターには 3D プリンタやレーザーカッターといった工作機械が使える FabSpace が設置され, 様々なバックグラウンドを持った人が毎日利用している. しかし, ユーザが欲しい 3D モデルをプリントするためには, まず CAD を使ってモデリングを行い, データを作成しなければならない. 3D データ共有サイトを利用して欲しい 3D モデルを探すという手段もあるが, 求めているものとぴったりの 3D モデルがなかったり, そもそもデータ自体が見つからなかったりすることもある. そうするとやはり CAD を使ってモデリングしたほうが早いこともある. もともと CAD は機械設計や建築を専門とする人に使われてきたものであり, 使いこなすにはある程度の学習が必要になる. いままで CAD を使ったことがない人にとってはこれが 3D プリントするまでの大きな壁となっている. 私自身も CAD を初めて使ったとき, 操作方法に慣れるのを苦労した経験がある. この壁をなるべく低くする一つの方法はより簡単な操作でモデリングをできるようにすることである. そこで, 本研究ではスケッチによる 3D モデリングシステムを開発する. このシステムは, ユーザがインターフェース上で絵を描くと, 入力された絵をもとに 3D モデルを生成する. 具体的には, Generative Adversarial Networks (GANs) を用いてスケッチからボクセルモデルの生成を行う. 本研究では二つの GAN を用意しボクセルとその色を生成する. 従来の Sketch-Based Modeling の研究ではアルゴリズムの性質上, 1 枚のスケッチに対して 1 個の 3D モデルが作られるようになっていたが, 本研究で提案する手法を用いれば 1 枚のスケッチに対して複数個の 3D モデルを生成できるようになる. この手法をもとにリアルタイムでスケッチからボクセルモデルに変換するインターフェースを実装した (4 節). 定性的・定量的評価に加え, 実験では生成した 3D モデルと人間が作った 3D モデルを並べ, どれが人間が作ったものかを実験参加者に当ててもらうテストを行った (5 節). テストの目的はどこまで人間の作った 3D モデルと判別のつかないものを生成できるかを見ることである. 結果, あるときは簡単に当てられてしまい, あるときは当てるのが困難となり成績は引き分けとなった. 最後に本研究に関する議論を 6 節にまとめた.

本研究の主な貢献をまとめると次のようになる.

- GAN を用いてスケッチからボクセルモデルとその色の生成を行った.
- 本研究の提案手法をもとに絵を描くとリアルタイムでボクセルモデルに変換するインターフェースを実装した.
- 一つの評価指標として, 生成した 3D モデルと人間が作った 3D モデルを並べ, 人間の作ったものを当ててもらう実験を行った.

2 Related Work

本節では関連研究について述べる．関連研究は多数存在する．ここでは混乱を避けるため関連研究を 3 つのグループに分けておく．1 つ目はスケッチによる 3D モデリングの研究であり, 2 つ目は本研究と研究目標は同じだがスケッチではない別のモデリング手法をとっている研究である．そして最後の 3 つ目は機械学習を用いてスケッチからデータ生成を行なっている研究である．以降ではこれら 3 つのグループの関連研究について順に述べていき, 本研究と関連研究で異なる点を明確化する．また, 本節の最後では関連研究と本研究との関係を図にまとめた (図 5).

2.1 Sketch-Based 3D Modeling

スケッチによる 3D モデリングの研究は今までも行われてきており, スケッチから 3D モデルに変換する手法はいくつか提案されてきた．例えば, SKETCH [51] と Teddy [20] がある．Teddy は入力されたスケッチを 3 次元のポリゴンモデルに変換する．ユーザは 3 次元形状の輪郭線を描くようにスケッチを行い, Teddy は描かれた輪郭線をもとにポリゴンのサーフェスモデルを作る．輪郭線から 3D モデルへ変換する他にも, 押し出し, 切り取り, 平滑化, 形状の変形（曲げる, 膨らます）など CAD の基本的な操作もスケッチから行えるようになっている．各機能の操作方法はスケッチの特性が活かされている．例えば球体の一部の領域から押し出しを行いたい場合, 押し出したい領域を閉曲線で示し, そこがどのように押し出されるのかを輪郭線で描くことによって操作できる．しかし, Teddy ではユーザが描いた線そのままをもとにして 3 次元形状を作っているため, 出力される 3D モデルはいつもキャラクターのような丸みを帯びた形状になっている[1)]．そのため, 車や飛行機, 建築物といった精密な 3D モデルを作成するのには向いていない．実際, Teddy は精密な 3D モデルを作るためではなく, おおよその形をすぐに作ることができるようにデザインされている．つまり, ラピッドプロトタイピングツールである．

これらの先行研究と本研究との違いは, 先行研究は確定的モデリングであるのに対して, 本研究は確率的モデリングである点である．確定的モデリングは同じスケッチを入力すれば決まったアルゴリズムに従って毎回同じ形が出てくる．一方で確率的モデリングは同じスケッチでも入力するたびに異なる形が出てくる．スケッチから 3D モデルに変換されるとき, それが確定的に行われるか確率的に行われるかが研究の大きな相違点である．そのため本研究のシステムはユーザが新しいアイデアを考えるときの支援にもなる．

[1)] 輪郭線の前処理は何も行われていないわけではなく, 3D モデルを作る前に輪郭線を平滑化する処理は行われている．しかし見た目には処理前と処理後の差はあまりない.

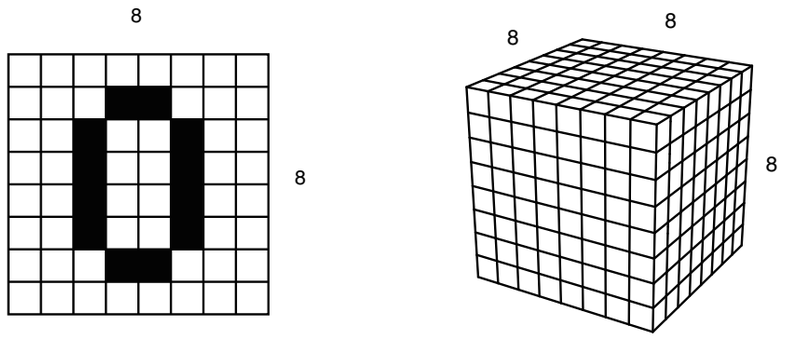


図 1　**ボクセルグリッド:** ボクセルはビットマップ画像を 3 次元に拡張したものと考えることができる．左は 8 × 8 のビットマップ画像の例．右は 8 × 8 × 8 のボクセルグリッド．ここでは見やすいように外側のグリッドだけ表示している．

2.2　3D Modeling Software for Beginners

2.1 節ではスケッチをもとにした 3D モデリングの研究について述べてきた．本節ではスケッチとは別のモデリング手法を採用している研究あるいは実際に開発されているモデリングソフトについて述べる．典型的な 3D モデリングは球体や立方体といったプリミティブと形状操作を組み合わせることで, 徐々に複雑な形を作っていくのが一般的である．昨今は 3D プリンタの普及も影響し, 従来の CAD とは異なるモデリングソフトがいくつか開発されている．それらをモデリング手法別に分類すると

- ボクセルによるモデリング
- パラメータ操作によるモデリング
- 彫刻によるモデリング

の 3 つに大別される．ボクセルによるモデリングとは, ある大きさの 3 次元グリッド上に立方体を配置していく方法である．この立方体を単位としてボクセルと呼んでいる．ボクセルはビットマップ画像を 3 次元に拡張したものと考えることができる (図 1)．例えば, グリッドの大きさは 3D モデルの解像度を表し, ビットマップ画像と同じようにグリッドが大きいほど解像度の高い 3D モデルを作ることができる．一般的な編集方法は画面のグリッド上でクリックまたはドラッグすることによってボクセルの追加・削除を行う．このボクセルモデリングが実装されているソフトウェアには Magicavoxel [12], Qubicle [30], VoxelShop [44] などがある．実際に Magicavoxel で作った 3D モデルを図 2 に示した．

2 つ目のパラメータ操作によるモデリングとは, 3D モデルに設定されているパラメータを変えることで望みの形を作っていく方法である．3D モデル自体はあらかじめ用意されており, ユーザは基本的にパラメータ操作を行う．そのためユーザは一からモデルを作る必要がなく, 1 つのモデルからたく

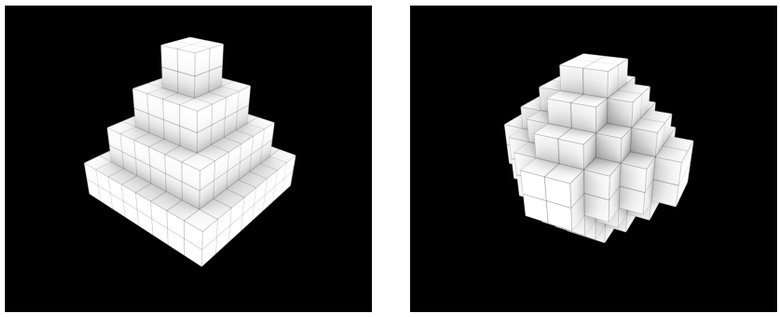


図 2　**ボクセルモデル:** Magicavoxel を使って $8 \times 8 \times 8$ のグリッド上でモデリングを行った例. これらは Magicavoxel のボクセル追加モードである”Attach”のみで作られた.

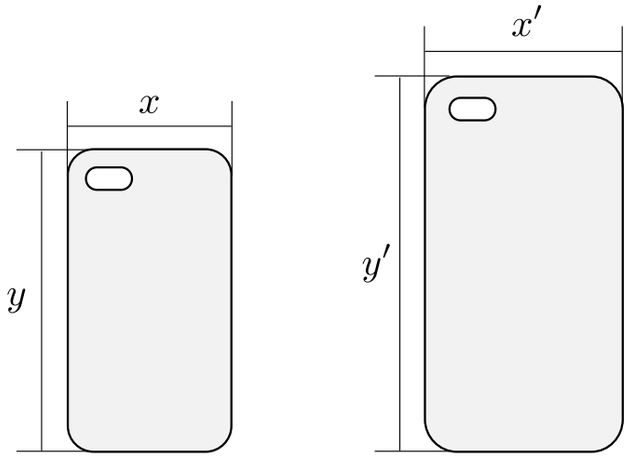


図 3　**パラメトリックモデリングの例:** A さんがお気に入りの携帯ケースを最近買い替えた新しい携帯でも使いたい場合, ケースの大きさをパラメトリックに変えることができたら, 同じケースを新しい携帯にも付けることができる. 例えば古い携帯の幅と高さが x, y(図左), 新しい携帯の幅と高さが x', y'(図右) でケース自体の厚みを考えなければ, ケースの幅と高さも x, y から x', y' に変えればよい.

さんの形状パターンを作り出せる. 例えば, A さんは自分の携帯にお気に入りのケースを使っているとする. ある日, A さんは携帯を買い替えたが, お気に入りのケースと携帯のサイズが合わなくなってしまった. ここでケースの大きさをパラメトリックに変えることができたらお気に入りのカバーを新しい携帯にも付けることができる (図 3). また, 既存の椅子を自分の体に合うように座面や背もたれの形を変えたい場合にも有効である. このパラメトリックモデリングが実装されている Web サイトあるいはソフトウェアには Vectary [42], Thingiverse [41], OpenSCAD [29] などがある. この内, Vectary と Thingiverse は Web 上で利用が可能であることから, 世界中の人たちが作った 3D モデルをカスタマイズする機能がある. それぞれの URL は参考文献に記した. OpenSCAD は他のモデリングソフトとは異なり, スクリプトを書くことによって 3D モデルを作るソフトウェアである. Thingiverse 上では, OpenSCAD のファイルをアップロードするとパラメータ操作ができるようになっている.

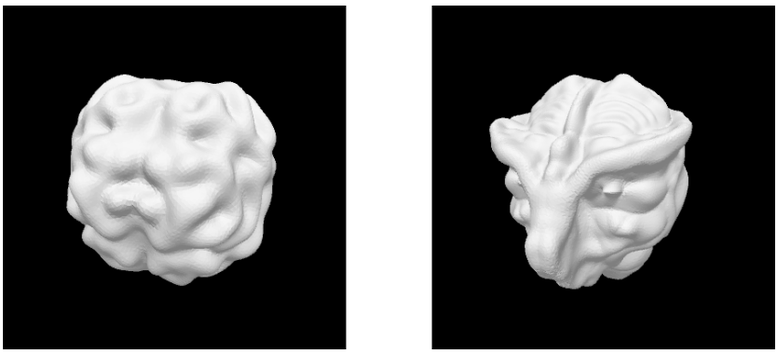


図 4　**スカルプトモデリングの例:** Sculptris を使って有機的な形をモデリングした．モデリング初心者でも簡単に使うことができ, 図の 3D モデルは私が初めてスカルプトモデリングをしたときのものである．

最後に彫刻によるモデリングとは, 粘土をこねてものを作るように, モデルを引き伸ばしたり削ったりすることで形を作っていく方法である．言い換えれば, コンピュータ上のバーチャル粘土をこねて 3D モデルを作る方法である．Rhinoceros[2)]や Solid Works[3)]などの CAD は車や飛行機といった精密な形を作ることに向いているのに対して, 彫刻によるモデリングは動物やモンスターといった有機的な形を作るのに向いている．もちろん生物の持っている肌や筋肉の細かい表現も可能である．このスカルプトモデリングが実装されているソフトウェアには Sculptris [36], Meshmixer [26], Blender [3] などがある．図 4 は Sculptris によって作られた 3D モデルである．

以上, 3 つのモデリング手法ごとにその特徴と具体的なソフトウェアについて述べてきた．これら 3 つのモデリング手法とは異なり, 本研究ではユーザに欲しい形の絵を描いてもらうことで 3D モデルを生成する．3D モデルの製作過程においてユーザに求める操作は画面上で絵を描くことのみである．そのため, 従来の CAD のように大量の操作ボタンをつける必要がなく, 他の手法と比べて UI を簡単にできるという利点がある．本研究がどのような形の作るのに向いているのかは 5 節で考察する．

2.3　3D Shape Generation and Retrieval

これまではスケッチによる 3D モデリングおよびスケッチではない別の 3D モデリング手法について述べてきた．本節では深層学習の生成モデルと機械学習を用いて二次元画像から三次元形状を生成している研究について述べる．

■ Deep Generative Models　機械学習分野において新たなデータを生成する研究は長い間行われてきた．データ生成の対象は画像, 音声, 文章, 三次元形状など多岐にわたる．特に深層学習 (Deep Learning) によってデータを生成する初期の研究は制限ボルツマンマシン (Restricted Boltzmann

2) https://www.rhino3d.com
3) http://www.solidworks.com

Machines; RBMs) [17] と深層信念ネットワーク (Deep Belief Networks: DBNs) [16] がある．最近においては変分自己符号化器 (Variational Auto-Encoders; VAEs) [10] と Generative Adversarial Networks (GANs) [14] がある[4)]．本研究ではこのうちの GAN を用いて開発を行う．GAN は学習アルゴリズムが比較的単純でありながら, 画像生成においては他の生成モデルよりも鮮明な画像を生成できるという特徴がある．もともと GAN には訓練時の学習が不安定であるという問題があったが, 学習を安定化し鮮明な画像を生成する方法が提案されている [31, 35]．他にも条件付き分布を学習させる方法 [13, 27] や生成器の分布と潜在変数の相互情報量を最大化する方法 [6], GAN にクラス分類をさせる方法 [28, 38] が提案されている．応用としては, 高解像度の画像生成 [19, 52, 53] や画像中の物体の操作 [32], 動画の生成 [43], 線描から画像の生成 [21, 54], 文章から画像の生成 [33], そして, CAD データを使ったボクセルモデルの生成 [47] がある．しかしながら, 文献 [47] では物体の形状しか生成していない．それに対して本研究では物体の形状に加えてその色も生成する．物体の形状と色を生成するにあたっては 2 種類の GAN を用いて形状を生成するステップと色を生成するステップの 2 段階に分けた．これは複数の GAN を積み上げたモデル [9, 19, 46, 52] を参考にした．深層学習と GAN についての詳細は 3 節で述べる．

■ 3D Shape Retrieval　上述のスケッチから三次元形状を新たに生成する方法に加えて, スケッチから三次元形状を得る方法はもうひとつある．それはスケッチと似ている 3D モデルをデータベースから検索してくる方法である．この方法では入力されたスケッチの特徴量を抽出し, 3D モデルの画像との類似度を計算していくことが一般的である．特徴量抽出には畳み込みニューラルネットワークを応用した研究 [45, 48, 49] やガボールフィルターを応用した研究 [11] がある．本研究の場合はデータベースを検索するのではなく三次元形状を新たに生成することに注意されたい．

■ 2D Image to 3D Shape　二次元画像から三次元形状を計算する問題はコンピュータビジョンにおける主要な研究のひとつである．単一の写真から計算する手法 [18, 22, 40, 50] や複数視点の写真から計算する手法 [23], 写真から物体の輪郭線を抽出し計算する手法 [15] など広範囲にわたって研究が行われてきた．最近では深層学習を用いて二次元画像からボクセルモデルを計算する研究もある．例えば, 3DR2N2[7] と 3D-GAN[47] がある．しかし, この 2 つの先行研究ではコンピュータビジョンの文脈に沿って写真からボクセルモデルを生成することに焦点が当てられていた．本研究では Sketch-Based 3D Modeling (2.1 節) の文脈に沿ってスケッチから 3D モデルを生成することに焦点を当てる．

[4)] 2016 年 12 月の時点ではまだ GAN の正式な日本語名はない．

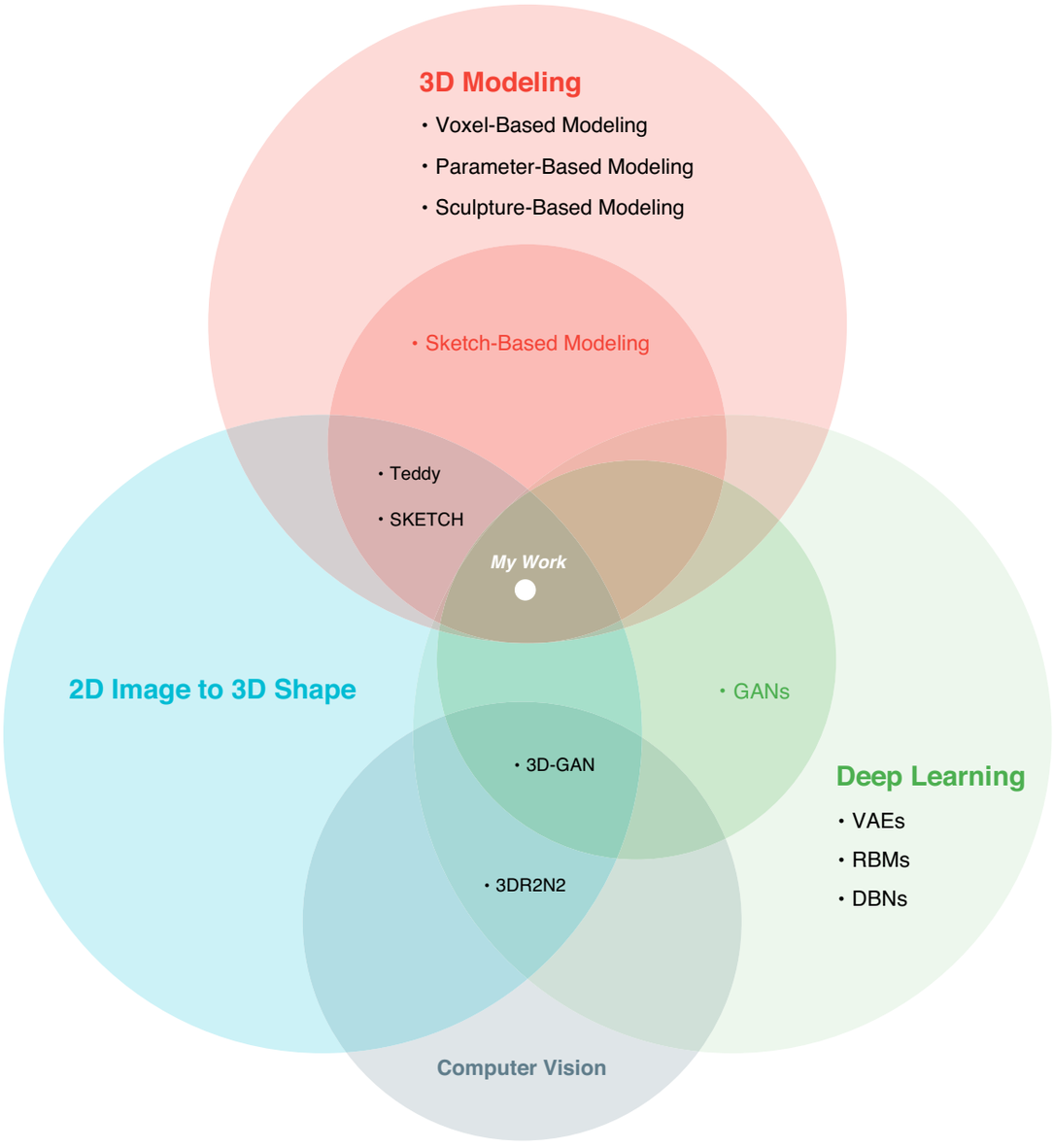


図 5　**本研究で取り組む領域:** 本研究の位置は白い点で示した.

3 Background

本節では開発のベースとなっている技術について具体的に説明する．1 節および 2 節でも述べてきたように本研究で用いる技術はニューラルネットワーク (以下ニューラルネット) である．深層学習とは, 多層構造のニューラルネットを用いた機械学習の総称である．深層学習と呼ばれているものには確率的に動作するネットワークモデルと確定的に動作するネットワークモデルの 2 つに分かれるが, 本節では後者のネットワークモデルについて述べる．まず, 3.1 節で確定的に動作する階層型ニューラルネットについて述べ, その次の 3.2 節では 2.3 節でも触れた GAN について述べる．

3.1 Neural Network

本研究でベースとなっている技術はニューラルネットである．ニューラルネットの研究は生物の神経回路における情報処理を数理モデルとして表そうとした試みに端を発する．例えば, 1943 年に提案された McCulloch-Pitts のニューロンモデルがある [25]．このようにしてニューラルネットは数学という人間に理解しやすい形で表されたものであるため, 元となった生物の神経回路を正確に記述しているかどうかの判断は難しい．そのため, 本節では神経科学的な観点からの議論は置いておき, 実用的な応用の観点から階層型ニューラルネットの数理モデルについて述べる．

3.1.1 Network Architecture

まずは入力信号 x に対して信号 y を出力する簡単なネットワーク (図 6) から始め, ニューラルネットの基本モジュールについて説明する．ここでは図 6 中の円をユニット, 円と円を結ぶ有向線をリンクと呼ぶことにする．ユニットをひとつの神経細胞, リンクを神経細胞同士をつなぐシナプスと考える．また, 特に断りがない限りこのような図は左から右へ信号が伝播していくものとする．さて, 図 6 のネットワークはどのように計算するのか．これは入力 x とリンク w の積と入力 1 とリンク b の積を足したものが y である．つまり,

$$y = wx + b$$

である．傾きを w, 切片を b として考えれば, これは 2 次元平面上の直線, 1 次関数になっている．w は重みと呼ばれる．b はバイアスと呼ばれ常に 1 を入力とする．生物の神経回路でいえば, 重みやバイアスは神経細胞同士をつなぐシナプスの結合強度である．

では次に入力信号が x_1 と x_2 の 2 つで信号 y を出力するネットワークを考える (図 7)．すると, y は

$$y = w_1x_1 + w_2x_2 + b$$

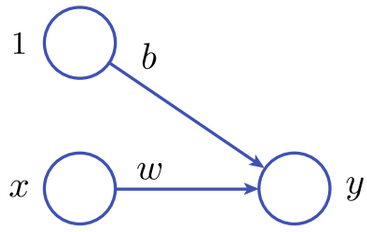


図 6　**ニューラルネット (1):** 入力 x に対して y を出力するネットワーク．このネットワークは $y = wx + b$ と計算する．左上のノードは常に 1 を入力とするバイアスと呼ばれているものである．

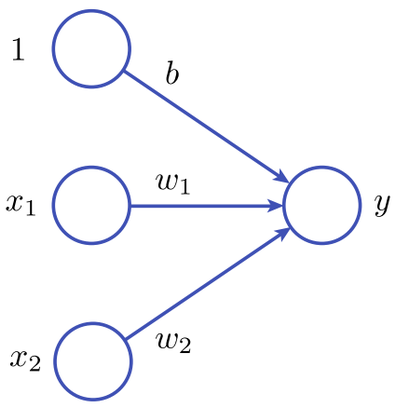


図 7　**ニューラルネット (2):** 2 つの入力 (x_1, x_2) に対して y を出力するネットワーク．このネットワークは $y = w_1x_1 + w_2x_2 + b$ と計算する．

となる．このようにして入力信号の数を 3 つ, 4 つ, 5 つ... と増やしていくと, 入力信号が n 個の場合 (図 8) の出力 y は

$$y = w_1x_1 + w_2x_2 + \cdots w_nx_n + b = \sum_{i=1}^{n} w_ix_i + b$$

のように表せる．すなわち, 各ノードの出力は, 複数の入力に対して, それぞれに重み w_i を掛け, それらをすべて足し合わせた値となる．いままでは出力の数は 1 つのままで入力の数のみを増やしてきたが, 出力側も同じようにノードを増やすことができる．例えば, 3 つの入力 (x_1, x_2, x_3) に対して 2 つの値 (y_1, y_2) を出力するネットワークは図 9 のようになる．入力 i 番目のノードから出力 j 番目のノードへつながっているリンクの重みを w_{ji} とすると, y_1 と y_2 はそれぞれ

$$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_2$$

$$y_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2$$

となる．この y_1 と y_2 は行列でまとめて計算することができる．すなわち

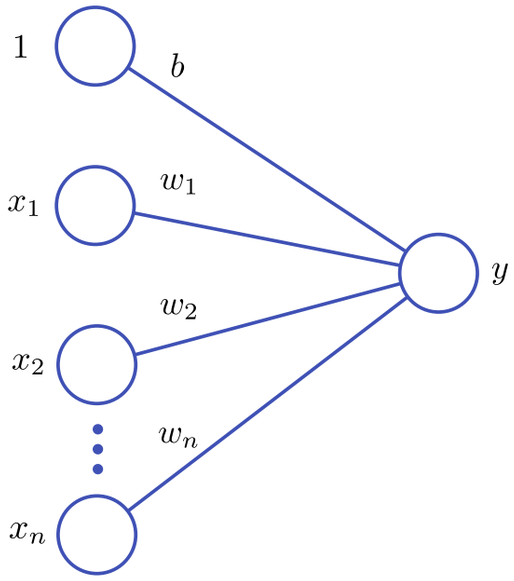


図 8　**ニューラルネット (3):** n 個の入力 $(x_1, x_2, \cdots, x_n)$ に対して y を出力するネットワーク. このネットワークは $y = \sum_{i=1}^{n} w_i x_i + b$ と計算する.

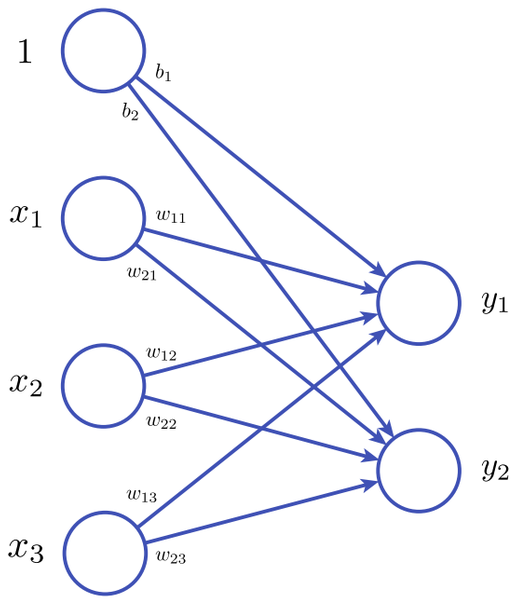


図 9　**ニューラルネット (4):** 3 つの入力 (x_1, x_2, x_3) に対して 2 つの値 (y_1, y_2) を出力するネットワーク. このネットワークは $y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1$, $y_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2$ と計算する.

$$\boldsymbol{y} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

太字の $\boldsymbol{y}$ はベクトル $(y_1, y_2)^{\mathrm{T}}$ である. ここまでは入力値と重みを掛けて足し合わせた値 $\boldsymbol{y}$ をネットワークの出力としていたが, 通常は $\boldsymbol{y}$ を変数とする活性化関数 f の値が出力となる. 活性化関数にはロジスティックシグモイド関数や双曲線正接関数などの非線形関数が用いられる.

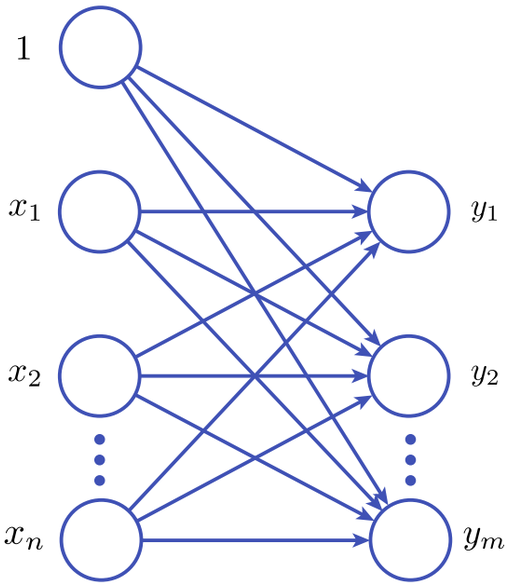


図 10　**ニューラルネット (5):** n 個の入力 $(x_1, x_2, \cdots, x_n)$ に対して m 個の値 $(y_1, y_2, \cdots, y_m)$ を出力するネットワーク. このネットワークの計算方法は式 (1) を参照していただきたい.

さて, 入力値が 3 つ, 出力値が 2 のネットワークを考えた. 一般化すると n 個の入力に対して m 個の値を出力するネットワークは図 10 のようになる. ネットワークの計算を行列で表せば

$$\boldsymbol{u} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{u}) = \begin{pmatrix} f(u_1) \\ f(u_2) \\ \vdots \\ f(u_m) \end{pmatrix} \tag{1}$$

となる. $\boldsymbol{f}$ は活性化関数である. ここまでが基本モジュールについての説明である. 図 10 のようなネットワークがニューラルネットの基本モジュールとなっているのである. 基本モジュールを何層にも積み重ねていくことで, ニューラルネットは構築されるのである.

実際にモジュールを 2 つ重ねたネットワークを図 11 に示した. このネットワークはベクトル $\boldsymbol{x} = (x_1, x_2)^{\mathrm{T}}$ を入力とし, ベクトル $\boldsymbol{y} = (y_1, y_2)^{\mathrm{T}}$ を出力する. 図中の縦に並ぶノードをまとめて層とすると, ネットワークは 3 つの層を持つニューラルネットとなる. 信号が入力される層を入力層, 中間を隠れ層, 出力される層を出力層と呼び, 入力層から順に 1 層, 2 層, 3 層と数えていく. ネット

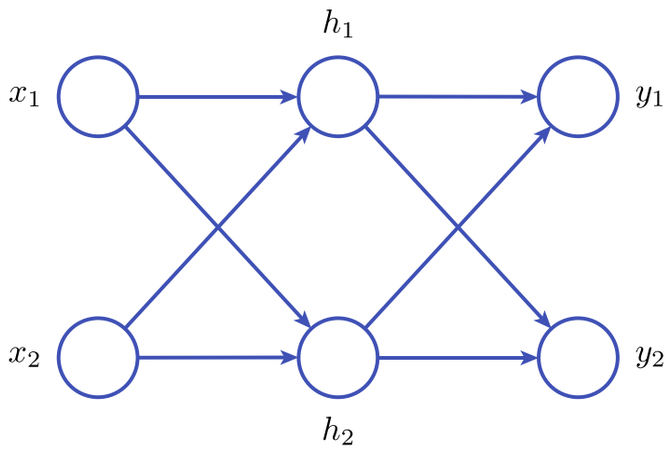


図 11　**ニューラルネット (6):** ニューラルネットでは図の縦に並ぶノードをまとめてひとつの層とする．したがって図の場合は 3 層のニューラルネットとなる．信号が入力される一番左の層を入力層, 真ん中の層を隠れ層 (または中間層), 信号が出力される一番右の層を出力層と呼ぶ．図中ではバイアスを省略している．

ワーク図 11 の入力から出力までは式 (1) の計算を重ねているモジュールの数だけ繰り返せばよいだけである．したがって, 2 回計算を繰り返す．第 L 層から出ている重みとバイアスをそれぞれ行列形式で $\boldsymbol{W}^{(L)}, \boldsymbol{b}^{(L)}$ と表すと, 計算手順は次のようになる．

$$\boldsymbol{h} = \boldsymbol{f}\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix}$$

$$\boldsymbol{y} = \boldsymbol{f}\left(\boldsymbol{W}^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}\right) = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}. \tag{2}$$

図 11 のニューラルネットは, このようにして入力から出力までの計算を行う．図のネットワーク構造はほんの一例にすぎない．後は自分の解きたい問題に対してノードや層の数を変え, 問題に適したネットワークを構築することができる．

入力信号 x に対して y を出力するネットワーク (図 6) から 3 層ネットワーク (図 11) までニューラルネットがどうのように信号を伝播させていくのかを見てきた．その結果, ニューラルネットは単純な掛け算と足し算を繰り返すことで計算できることが分かった．しかしながら, このままではニューラルネットは使い物にならない．重みとバイアスをランダムに設定しているだけではあまり意味のない値を出力するだけである．ニューラルネットを使えるものにするためには与えられた問題に対して, 重みとバイアスを最適な値に設定する必要がある．次節では重みとバイアスの最適値を求める方法について述べる．

図 12　**データセットの例:** ( ) 内の左はスケッチ, 右はボクセルモデル. 訓練時にはこのペアを数千から数万個用意し学習させる.

3.1.2　Training Network

3.1.1 節ではニューラルネットの構造と伝播の計算方法について述べた. 実際に使えるニューラルネットを得るまでにはネットワーク構造の設計とパラメータ最適化という 2 つのステップある. ネットワーク構造は前節で述べた通りである. 本節ではパラメータである重みとバイアスの最適値を求める方法について述べる. 以降ではすべての重みとバイアスをまとめてパラメータ $\boldsymbol{\theta}$ として表すことにする.

最適値を求めるためには, まず解きたい問題を設定する必要がある. 本研究の場合はスケッチ画像 $\boldsymbol{x}$ から適切なボクセルモデル $\boldsymbol{y}$ を出力する関数を求めたい. そのため, スケッチ画像を入力にとり, ボクセルモデルを出力する問題を例に考える. いまスケッチ画像とボクセルモデルが対応づけられたペアが N 個あるとする: $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \cdots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$ (図 12). ここでの問題はなるべく多くのスケッチに対して正しいボクセルモデルを出力できるようにパラメータ $\boldsymbol{\theta}$ を調整していくことである. 例えば, $\boldsymbol{x}_1$ が入力されたならば対応する $\boldsymbol{y}_1$ に, $\boldsymbol{x}_2$ が入力されたならば $\boldsymbol{y}_2$ に近いボクセルモデルを出力するように調整するのである. この調整作業を訓練あるいは学習という. 具体的な訓練手順は次の通りである.

1. データセットを用意する.
2. 解きたい問題に応じて損失関数を設定する.
3. 損失関数の勾配を求めパラメータ $\boldsymbol{\theta}$ を調整する.

1. データセットを用意する. データセットは, 例えば, スケッチ画像とボクセルモデルが対応づけられた N 個のペアである.

2. 解きたい問題に応じて損失関数を設定する. 損失関数とはニューラルネットの出力値と実際のボクセルモデルとの誤差を測る関数である. ニューラルネットの学習において, 通常, 出力が連続値の場合は二乗誤差, 出力が 2 値の場合はクロスエントロピー誤差が損失関数として用いられる. ボクセルグリッド上ではボクセルがない状態を 0, ある状態を 1 とすればボクセルデータは 2 値で表すことができるので損失関数にはクロスエントロピー誤差,

$$L = -\sum_{i=1}^{m} y_i \log y'_i + (1 - y_i) \log (1 - y'_i),$$

を用いることになる．y'_i はニューラルネットの出力 $(y'_1, y'_2, \cdots, y'_m)$ の i 番目の要素である．

3．損失関数の勾配を求めパラメータ θ を調整する． 2. で与えた損失関数を小さくするように $\boldsymbol{\theta}$ を調整する．言い換えれば, ニューラルネットの出力値とボクセルモデルとの誤差を小さくするように $\boldsymbol{\theta}$ を調整するのである．調整には勾配降下法がよく利用される．勾配降下法はその名前の通り, 損失関数の $\boldsymbol{\theta}$ における勾配を求め, 勾配を下っていく方向に $\boldsymbol{\theta}$ を調整する．例えば, L 層の i 番目のノードから $L+1$ 層の j 番目のノードをつなぐリンクの重み $w_{ji}^{(L)}$ を調整したい場合は,

$$w_{ji}^{(L)} \leftarrow w_{ji}^{(L)} - \alpha \frac{\partial L}{\partial w_{ji}^{(L)}}$$

と更新する．$\leftarrow$ は $w_{ji}^{(L)}$ を $w_{ji}^{(L)} - \alpha \frac{\partial L}{\partial w_{ji}^{(L)}}$ に更新することを意味する．α は学習率で 1 より小さな値を取ることが多い．これを, データセットを入力する度に, すべてのパラメータ $\boldsymbol{\theta}$ について繰り返し行うことで最適値を求めることができる．しかし, 3.1.1 節の図 6 のようなパラメータが少ないネットワークならばひとつひとつ勾配を計算しても問題はないが, パラメータ数が数千から数万に増えた場合に勾配をひとつひとつ計算していたら大変な時間がかかってしまうという問題がある．勾配を効率よく求める方法には誤差逆伝播法がある．詳しくは文献 [34] を参照していただきたい．

3.2 Generative Adversarial Nets

本節では開発のベースとなっている Generative Adversarial Nets (GANs) [14] について述べる．GAN には Generator (以下 G) と Discriminator (以下 D) の 2 つのネットワークが存在する (図 13)．G は分布 p_z からサンプルした乱数 $\boldsymbol{z}$ を入力としデータ $\boldsymbol{x}_g$ を出力する生成器である: $\boldsymbol{x}_g = G(\boldsymbol{z})$．乱数 $\boldsymbol{z}$ は一様分布や正規分布からサンプルするのが一般的である．一方, D は G によって生成されたデータ $\boldsymbol{x}_g$ と訓練データ $\boldsymbol{x}_{data}$ を入力とし両者を判別する分類器である．D は入力されたデータ X が訓練データ由来である確率を出力する: $P(Y = data|X) = D(X)$.

D は $P(Y = data|X = \boldsymbol{x}_{data})$ を最大にし $P(Y = data|X = \boldsymbol{x}_g)$ を最小にするように訓練する．つまり, 正しい判別ができるように訓練する．それとは反対に G は $P(Y = data|X = \boldsymbol{x}_g)$ を最大にするように訓練する．つまり, D が分類を間違えるように G を訓練するのである．目標は, このように D と G の訓練を交互に行っていくことで, 訓練データと見分けがつかないデータを生成する G を得ることである．具体的には, 次の目的関数を G は最小化, D は最大化するように最適化する:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}\left[\log D(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim p_z}\left[\log\left(1 - D(G(\boldsymbol{z}))\right)\right]. \tag{3}$$

$\mathbb{E}[\cdot]$ は [ ] 内の期待値を意味する．$\boldsymbol{x}$ は訓練データであり，p_{data} はそのデータ分布である．p_g を $G(z)$ のデータ分布とすれば，式 (3) は $p_g = p_{data}$ のときに大域的最適解を持つ．よって GAN は生成器 G に分布 p_{data} を学習させる手法だと考えることもできる．実際に G を訓練する場合は式 (3) を最小化するのではなく，別の目的関数 $\mathbb{E}_{\boldsymbol{z}\sim p_z}[\log D(G(\boldsymbol{z}))]$ を最大化するように最適化する．学習の初期段階では G が生成するデータは訓練データと明らかに異なり，十分な勾配が得られないためである．これらの目的関数は符号を反転すれば，3.1.2 節で述べた損失関数となり，勾配降下法によって最適化することができる．

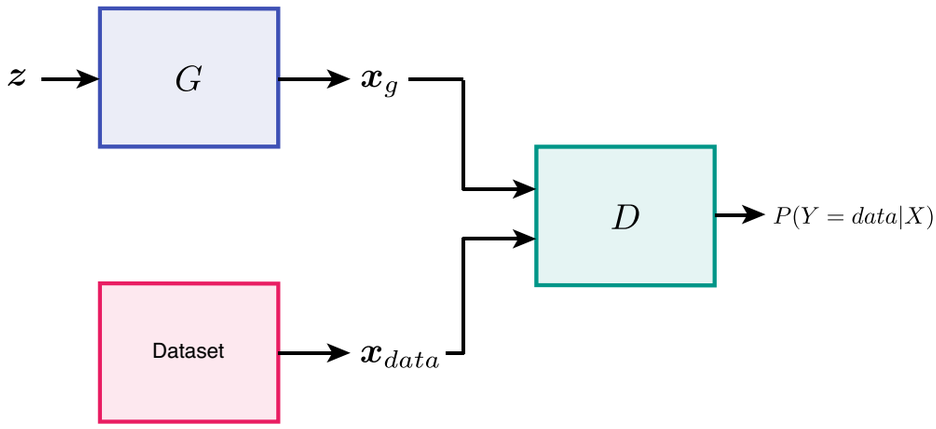


図 13　**Generative Adversarial Nets:** G と書かれた青色のボックスは生成器，D と書かれた緑色のボックスは分類器である．赤色のボックスは訓練データ．G は乱数 $\boldsymbol{z}$ を入力としてデータ $\boldsymbol{x}_g$ を出力する．目標は訓練データと見分けがつかないデータを G に生成させることである．これは生成データと訓練データを正しく判別できるように D を訓練し，D が判別を間違えるように G を訓練することで達成される．

4 Method

本節では提案する具体的な手法について述べる．1 節でも述べたように本研究では GAN をベースとして 3D モデリングシステムを開発する．このシステムの最終的な出力物は彩色されたボクセルモデルとなる．なお, GAN についての詳細は 3.2 節で述べた通りである．本節の構造は以下のようになっている．4.1 節では入力されたスケッチからどのようにボクセルモデルを生成するのか, その具体的なアルゴリズムについて述べる．4.2 節では本研究で実装したソフトウェアについて, 実際にユーザがどのように使うのかを述べる．4.3 節では実装の詳細を述べる．

次の節に行く前に色付きボクセルモデルの表し方について簡単に述べておく．色付きボクセルモデルは $d \times h \times w \times 4$ のテンソルとして表される．ここで d は深さ, h は高さ, w は幅, 4 はチャンネル数である．4 チャンネルの内訳は 1 チャンネル分がボクセル, 残りの 3 チャンネル分が RGB の色情報となっている (図 14).

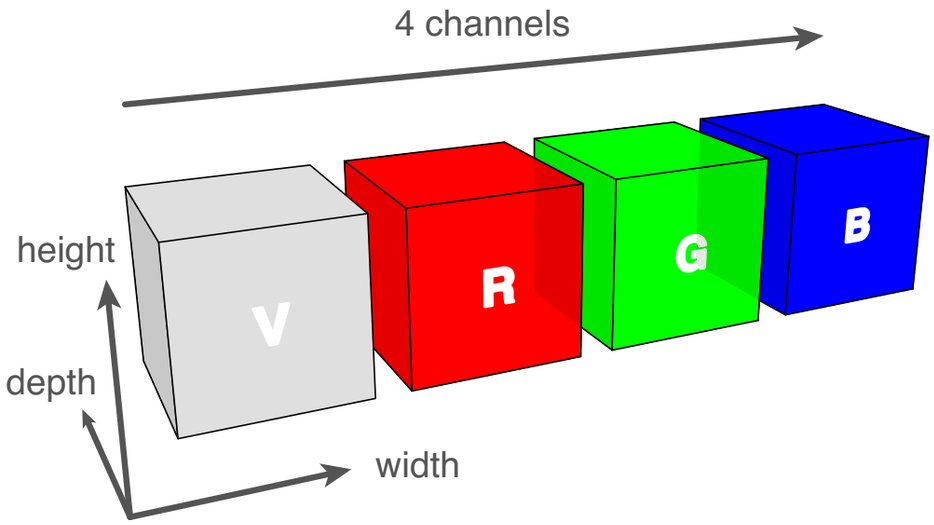


図 14　**色付きボクセルモデルの表し方:** 図の立方体は三次元のボクセルグリッドを表す．一番左の立方体はボクセル (マスクと言った方がよいか) を格納するグリッド．残り 3 つの立方体はそれぞれ赤 (R), 緑 (G), 青 (B) の色情報を格納するグリッドになっている．このようにして色付きボクセルモデルは 4 次元の配列として表される．

4.1 Algorithm

入力されたスケッチからどのようにボクセルモデルが生成されるのか．まずは図 15 のネットワーク構造をもとにしながら説明する．図 15 に示したように提案する手法には 2 種類の生成器が存在する．1 つ目は, スケッチ画像からボクセルを生成する G_v である．2 つ目は, 生成されたボクセルから各ボ

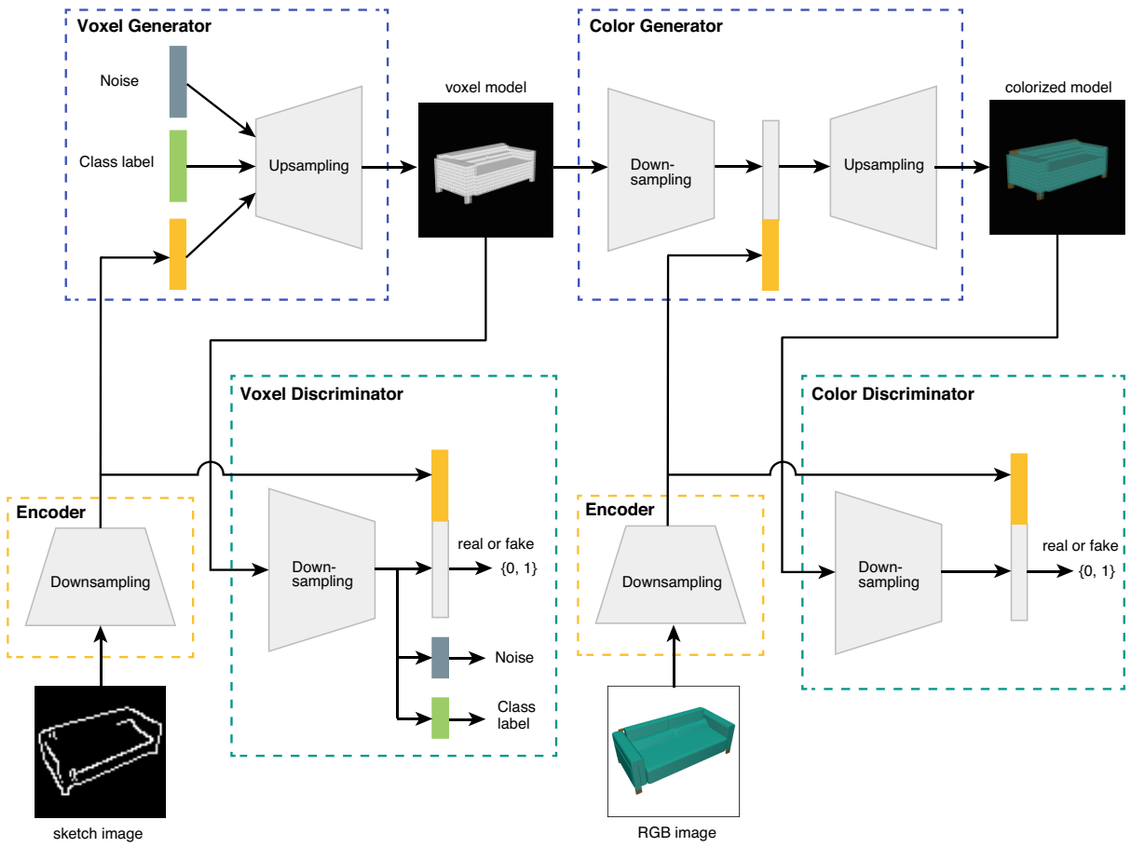


図 15　Model Architecture.

クセルの色を生成する G_c である[5]. G_v と G_c はそれぞれ分類器 D_v と D_c を用意し, GAN のアルゴリズムに従って別々に訓練を行う. よってスケッチ画像からボクセルモデルが生成されるまでの工程には以下の 2 つのステップがある.

1. **ボクセルの生成:** スケッチ画像 $\boldsymbol{s}$, 一様分布からサンプルされた乱数 $\boldsymbol{z}$, クラスラベル $\boldsymbol{c}$ の 3 つを入力にとり, それらの情報をもとにボクセル $\boldsymbol{x}_v$ を生成する. この段階ではまだ色は付いていないことに注意されたい.
2. **色の生成:** 1. で生成されたボクセルモデルと RGB 画像から各ボクセルの色 $\boldsymbol{x}_c$ を RGB 形式で生成する. ただし, $\boldsymbol{x}_c \in \mathbb{R}^{d \times h \times w \times 3}$. RGB 画像は着色を行う際の参考として入力される. 訓練時はレンダリング画像を入力する.

最終的な出力は第 1 ステップで生成されたボクセル $\boldsymbol{x}_v$ と第 2 ステップで生成された各ボクセルの色 $\boldsymbol{x}_c$ を合わせたものになる. そのためユーザはスケッチに色情報も付け加えることができる. 各ステップのデータ生成には GAN を別々に訓練する. 次に訓練方法とネットワーク構造の詳細を述べる.

[5] G_v と G_c の v と c は *voxel* と *color* から頭文字をとった.

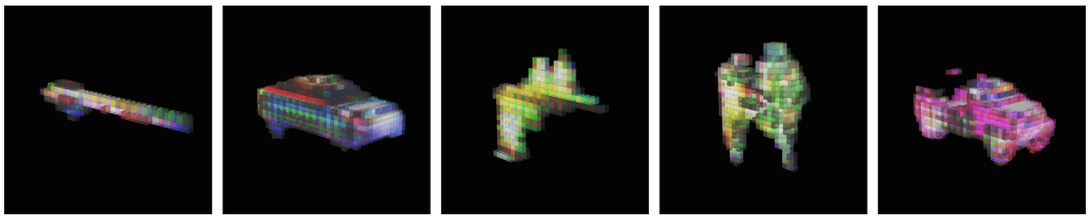


図 16　**学習がうまくいかなかった例:** ニューラルネットにボクセルと色を同時に生成させようとしたとき, 色がうまく生成されなくなってしまった例. そこで本研究ではボクセルと色を分けて生成させることにした.

4.1.1　Voxel Data Generation

最終的に生成したいデータは色付きのボクセルモデルであるが, 本研究ではボクセルと色を一度に生成することはせず, まず最初にボクセル $\boldsymbol{x}_v$ のみを生成することから始める. なぜかというと, ボクセルモデルとその色を同時に生成しようとするとタスクが難しくなり学習が進まなくなってしまうということが実験的に分かったからである. 同時に生成させようとすると図 16 のように様々な色が入り混じったようなモデルが出てくるようになってしまった. そこで, タスクをボクセル生成と色生成の 2 つに分けることで, うまく学習させることができないか試みた. 以降ではボクセル生成器 G_v と分類器 D_v の具体的な訓練方法とネットワーク構造について述べる.

■ 訓練方法　分類器 D_v と生成器 G_v はそれぞれ次の損失関数を最小化することで訓練する:

$$\mathcal{L}_{D_v} = \mathcal{L}_{D_v}^{adv} + \mathcal{L}_{D_v}^{cls} + \mathcal{L}^{ent} \tag{4}$$

$$\mathcal{L}_{G_v} = \mathcal{L}_{G_v}^{adv} + \mathcal{L}_{G_v}^{cls} + \mathcal{L}^{ent}. \tag{5}$$

ここで

$$\mathcal{L}_{D_v}^{adv} = \mathbb{E}_{(\boldsymbol{x}_0,\boldsymbol{s})\sim p_{data}}[-\log D_v(\boldsymbol{x}_0,\boldsymbol{s})] + \mathbb{E}_{(\boldsymbol{s},\boldsymbol{c})\sim p_{data},\ \boldsymbol{z}\sim p_z}[-\log\left(1 - D_v(G_v(\boldsymbol{z},\boldsymbol{s},\boldsymbol{c}),\boldsymbol{s})\right)] \tag{6}$$

$$\mathcal{L}_{G_v}^{adv} = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{c})\sim p_{data},\ \boldsymbol{z}\sim p_z}[-\log D_v(G_v(\boldsymbol{z},\boldsymbol{s},\boldsymbol{c}),\boldsymbol{s})]. \tag{7}$$

$\boldsymbol{z}$ は分布 p_z からサンプルする乱数 (本研究では一様分布から得る). $\boldsymbol{x}_0$ は訓練データのボクセルモデル. $\boldsymbol{s}$ はスケッチ画像. $\boldsymbol{c}$ は「椅子」「車」といったクラスラベルである. クラスラベルは one-hot ベクトルとして表す. One-hot ベクトルとはある 1 つの要素だけが 1 で, その他は 0 であるベクトルのことである. 例: $(0,0,1,0,0,0)$. 各次元がそのクラスか否かを表す. よってこのベクトルの次元数はデータセットのクラス数と等しい. $\boldsymbol{s},\boldsymbol{c}$ も訓練データから得る. $\mathcal{L}_{D_v}^{cls}$ と $\mathcal{L}_{G_v}^{cls}$ は通常のクラス分類を学習させるときに用いるクロスエントロピー:

$$\mathcal{L}_{D_v}^{cls} = \mathbb{E}_{\boldsymbol{x}_0 \sim p_{data}}[-\log D_v(\boldsymbol{x}_0)]$$

$$\mathcal{L}_{G_v}^{cls} = \mathbb{E}_{\boldsymbol{x}_v \sim p_g}[-\log D_v(\boldsymbol{x}_v)]$$

である．ここで p_g は生成器 G_v の分布である．また, D_v の出力は式 (6), (7) のときとは異なり, C をクラスラベルとしたときの条件付き確率 $P(C|X) = D(X)$ となっていることに注意されたい．クラス分類も学習させる手法は [28] で提案されている．先行研究では GAN を用いて画像を生成したときに, 本物のような画像は生成されるが, それが「犬」なのか「人」なのか, それともももっと別のものなのかよく分からない画像が生成されることがあった．クラスラベルを入れることによって, 判別可能で全体としてまとまりのあるデータを生成できるようになることが期待される．最後に $\mathcal{L}^{ent}$ は [19] で提案されている entropy loss:

$$\mathcal{L}^{ent} = \mathbb{E}_{\boldsymbol{z} \sim p_z}[\mathbb{E}_{\boldsymbol{x}_v \sim p_g}[-\log Q(\boldsymbol{z}|\boldsymbol{x}_v)]]$$

である．Q は真の事後分布 $P(\boldsymbol{z}|\boldsymbol{x}_v)$ を近似する関数であり, ニューラルネットによってパラメータ化する．Q は出力層以外のパラメータ (畳み込み層) はすべて分類器 D_v と共有している．GAN の学習において, 乱数 $\boldsymbol{z}$ に加えて画像やクラスラベルなどの条件も入力して学習させると, 乱数 $\boldsymbol{z}$ を無視し画像やクラスラベルから確定的にデータを生成するようになってしまうという問題がある．しかし, entropy loss を導入することでこの問題を回避することができる．Entropy loss の考え方はスケッチ $\boldsymbol{s}$ といった条件が与えられたもとでの $\boldsymbol{x}_v$ の条件付きエントロピー $H[\boldsymbol{x}_v|\boldsymbol{s}]$ をできるだけ大きくすることで $\boldsymbol{x}_v$ に多様性を持たせようというものである．情報理論におけるエントロピー $H[X]$ とは事象 X の中でどれが発生したかの不確定度を表し, 条件付きエントロピー $H[X|Y]$ とは Y が何であるか知ったもとでの X の不確定度を表す．つまり, $H[\boldsymbol{x}_v|\boldsymbol{s}]$ はスケッチ $\boldsymbol{s}$ が入力されたとき $\boldsymbol{x}_v$ が何であるかの不確定度を表す．この不確定度を大きくすれば, 乱数 $\boldsymbol{z}$ が無視されてしまう問題を回避できそうである．先ほどの $\mathcal{L}^{ent}$ を小さくすることは $H[\boldsymbol{x}_v|\boldsymbol{s}]$ を大きくすることに相当するため, 実際には $H[\boldsymbol{x}_v|\boldsymbol{s}]$ よりも扱いやすい $\mathcal{L}^{ent}$ を損失関数として導入している．証明は [19] を参照．学習後にデータを生成する際には, 乱数は一様分布からサンプルし, クラスラベルは明示的に入力する．例えば, 「車」を生成したい場合は「車」のクラスラベルを入力する．

■ ネットワーク構造　生成器 G_v は次のような構造になっている．図 15 に示したように, スケッチ画像は 2 次元畳み込みニューラルネットによって次元を圧縮し特徴量 $\boldsymbol{s}'$ へと変換する．その後, $\boldsymbol{s}'$ を乱数 $\boldsymbol{z}$ とクラスラベル $\boldsymbol{c}$ と連結し, G_v へ入力する．最後は 3 次元逆畳み込みニューラルネットによって任意のボクセルサイズまで復号化する．一方, 分類器 D_v は次のような構造になっている．入力されたボクセルモデルを 3 次元畳み込みニューラルネットによって符号化した後, それをスケッチ

の特徴量 $\boldsymbol{s}'$ と連結し, 最後に全結合層によってボクセルモデルが本物 (データセット) である確率を出力する.

4.1.2 Colorized Voxel Generation

色生成器 G_c と分類器 D_c の訓練は G_v を学習させた後に行う. 第 2 ステップでの目標は第 1 ステップで生成されたボクセルモデルに適切な着色を行うことである. 以下にその訓練方法とネットワーク構造について述べる.

■ 訓練方法 分類器 D_c と生成器 G_c はそれぞれ次の損失関数を最小化することで訓練する:

$$\mathcal{L}_{D_c} = \mathbb{E}_{(\boldsymbol{x}_1,\boldsymbol{r})\sim p_{data}}[-\log D_c(\boldsymbol{x}_1,\boldsymbol{r})] + \mathbb{E}_{\boldsymbol{r}\sim p_{data},\, \boldsymbol{x}_v\sim p_g}[-\log\left(1 - D_c(G_c(\boldsymbol{x}_v,\boldsymbol{r}),\boldsymbol{r})\right)] \tag{8}$$

$$\mathcal{L}_{G_c} = \mathbb{E}_{\boldsymbol{r}\sim p_{data},\, \boldsymbol{x}_v\sim p_g}[-\log D_c(G_c(\boldsymbol{x}_v,\boldsymbol{r}),\boldsymbol{r})] + \mathcal{L}_{G_c}^{L1}. \tag{9}$$

ここで $\mathcal{L}_{G_c}^{L1}$ はボクセルの色情報 (正解データ) $\boldsymbol{x}_1$ と G_c の差の L1 ノルムである:

$$\mathcal{L}_{G_c}^{L1} = \mathbb{E}_{(\boldsymbol{x}_1,\boldsymbol{r})\sim p_{data},\, \boldsymbol{x}_v\sim p_g}[\|\boldsymbol{x}_1 - G_c(\boldsymbol{x}_v,\boldsymbol{r})\|_1].$$

この L1 ノルムを導入することによって, GAN が作るデータ中のごみを減らすことが期待される. $\boldsymbol{r}$ はレンダリング画像. $\boldsymbol{x}_v$ は第 1 ステップで生成されたボクセルモデルである. p_g は生成器 G_v の分布である.

■ ネットワーク構造 生成器 G_c は次のような構造になっている. まず, レンダリング画像 $\boldsymbol{r}$ を 2 次元畳み込みニューラルネットによって符号化する. 同時に, ボクセルモデルも 3 次元畳み込みニューラルネットによって符号化する. 次に符号化したレンダリング画像とボクセルモデルを連結し, 3 次元逆畳み込みニューラルネットによって第 1 ステップのボクセルモデルと同じ大きさまで復号化する. 一方, 分類器 D_c の構造は第 1 ステップの D_v と同じ構造になっている. 異なるのは入力する画像がスケッチではなく RGB 画像になっている点である.

4.2 User Interface

4.1 節で述べた手法をもとに, 絵を描くと瞬時にボクセルモデルを生成するインターフェースを開発した (図 17, 18). このインターフェースは iGAN[54] を参考にしている. 具体的にはユーザが図 17 の右側にあるキャンバスに絵を描くと, 左側の画面にボクセルモデルが表示されるようになっている. 線を描くとすぐにボクセルモデルが表示されるため, ユーザは絵を描きながらモデリングを行うことになる. ペンは線を描くモードと色を塗るモードの 2 種類ある. 色はマウスの右クリックで出てくるパレットから変えられる. この 2 つのモードを使い分けることで形を作ったり色を付けたりする. ま

た, 画面左下の保存ボタンを押すと OBJ 形式でボクセルモデルが保存される.

4.3 Implementation

実装にあたってプログラミング言語は Python を使用した. ニューラルネットの計算には機械学習の計算ライブラリである TensorFlow[2] を使用した. インターフェースは PyQt[1] を使って開発した. また, OS は Ubuntu 14.04.4 LTS を使用し, *NVIDIA GeForce GTX TITAN X* という GPU で計算を行った. もちろん, Mac や Windows からでもインターフェースの実行はできる. Mac の OS X Mavericks 10.9.5 上では動作確認済みである. すべてのソースコードは次の URL から見ることができる: https://github.com/maxorange/pix2vox.

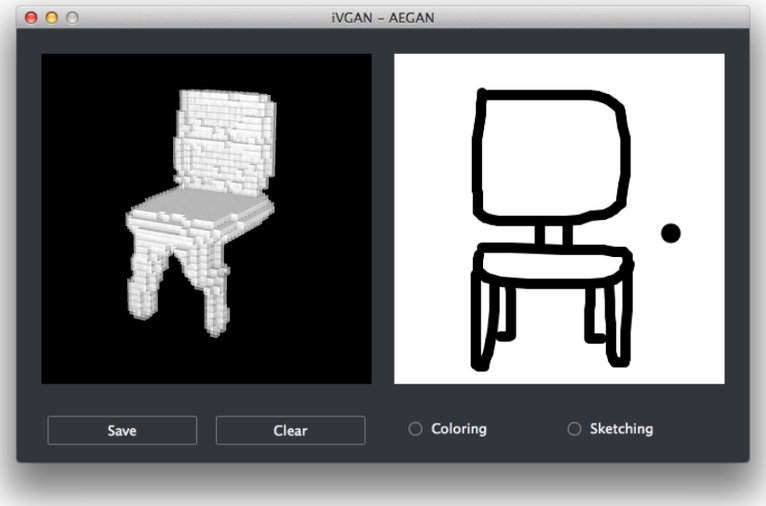


図 17 **開発したインターフェース:** 画面右のキャンバスに絵を描くと画面左にボクセルモデルが表示される. キャンバス上の黒い点がペンである. 詳細は 4.2 節を参照.

図 18　**ボクセルモデルができてくる様子**

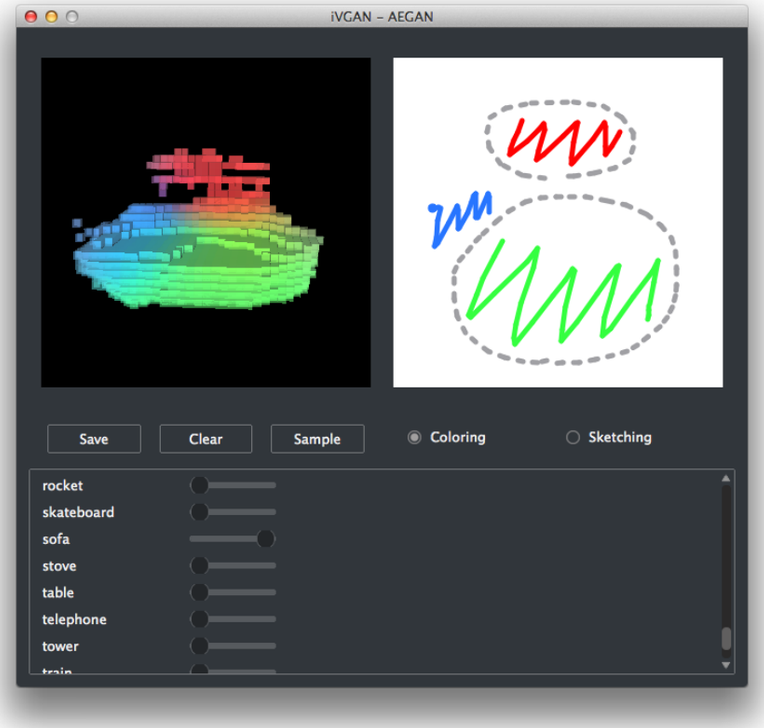


図 19　**色付きボクセルモデルの生成:** ”Sample”ボタンは新しく乱数 z をサンプルする．画面下のボックスではバーを調整することでクラスラベルを指定することができる．詳細は 5.2 を参照．

5 Experiments

本節では手法についての定性的・定量的評価とユーザ実験について述べる．

■ Dataset　全体の実験を通じて，訓練用のデータセットは ShapeNet [5] のサブセットである ShapeNetCore を使用する．ShapeNetCore に含まれるデータの例を図 21 に示した．ShapeNetCore には「椅子」や「車」「飛行機」など全部で 57 のクラスがある．それぞれのクラスには数百から数千個の CAD データがある．CAD データの数は合計で約 57,000 個ある．この CAD データは OBJ ファイルとして提供されていたため，すべてのデータに対してカラーボクセル化を行った．ただし，サイズは $32 \times 32 \times 32$ とし，モデルの中身もボクセルで埋めるようにした (Solid Voxelization)．訓練時に用いる RGB 画像は 1 つの CAD データにつき 5 つの異なる視点からレンダリングしたものを使う (全部で約 285,000 枚の画像)．5 つの視点は 3D モデルの中心点から視点までの距離は一定という制約条件の下でランダムに選んだ．また，スケッチ画像には Canny のエッジ検出法 [4] を用いてレンダリング画像から輪郭線を抽出したものを使う．レンダリング画像とエッジ検出した画像は画面いっぱいに物体が配置されるよう余白の切り取りを行い，サイズは 64×64 にそろえる．以上のカラーボクセル化，レンダリング，エッジ検出したものを図 20 に示す．訓練時にはこれら 3 種類のデータが 1 つのセットとなる．

■ Training　訓練時における最適化には Adam[24] を使用する．学習率は，第 1 ステップの生成器は 0.0004，分類器は 0.0001 とする．第 2 ステップの生成器は 0.0002，分類器は 0.0001 とする．β_1 は [31] にならって，すべて 0.5 に設定する．ミニバッチサイズは第 1 ステップは 64，第 2 ステップは 32 にする．それぞれのステップにつき訓練は約 12 時間かけて行った．いま述べた設定値とネットワークの具体的なハイパーパラメータは付録 A にまとめてある．

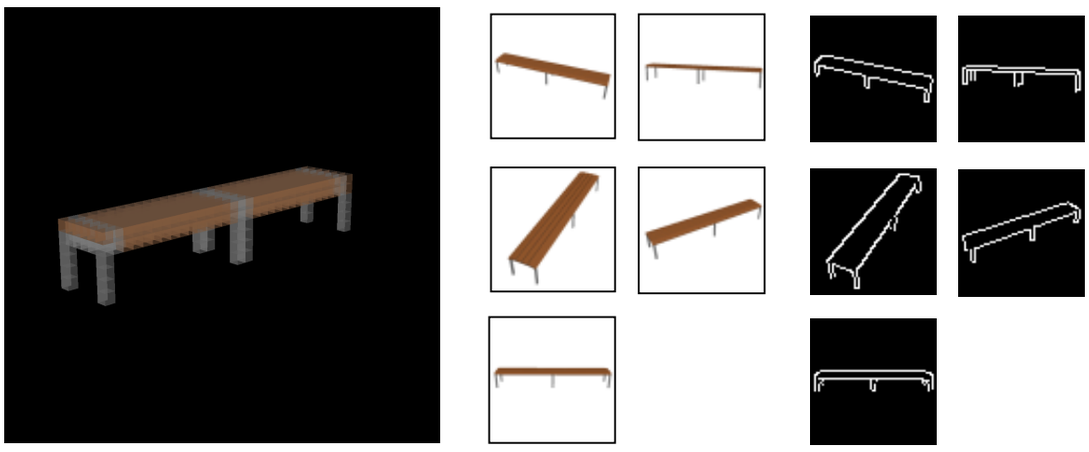


図 20　**訓練データ:** 左から順にボクセルモデル，レンダリング画像，エッジ検出画像．ボクセルサイズは $32 \times 32 \times 32$，画像サイズは 64×64 である．

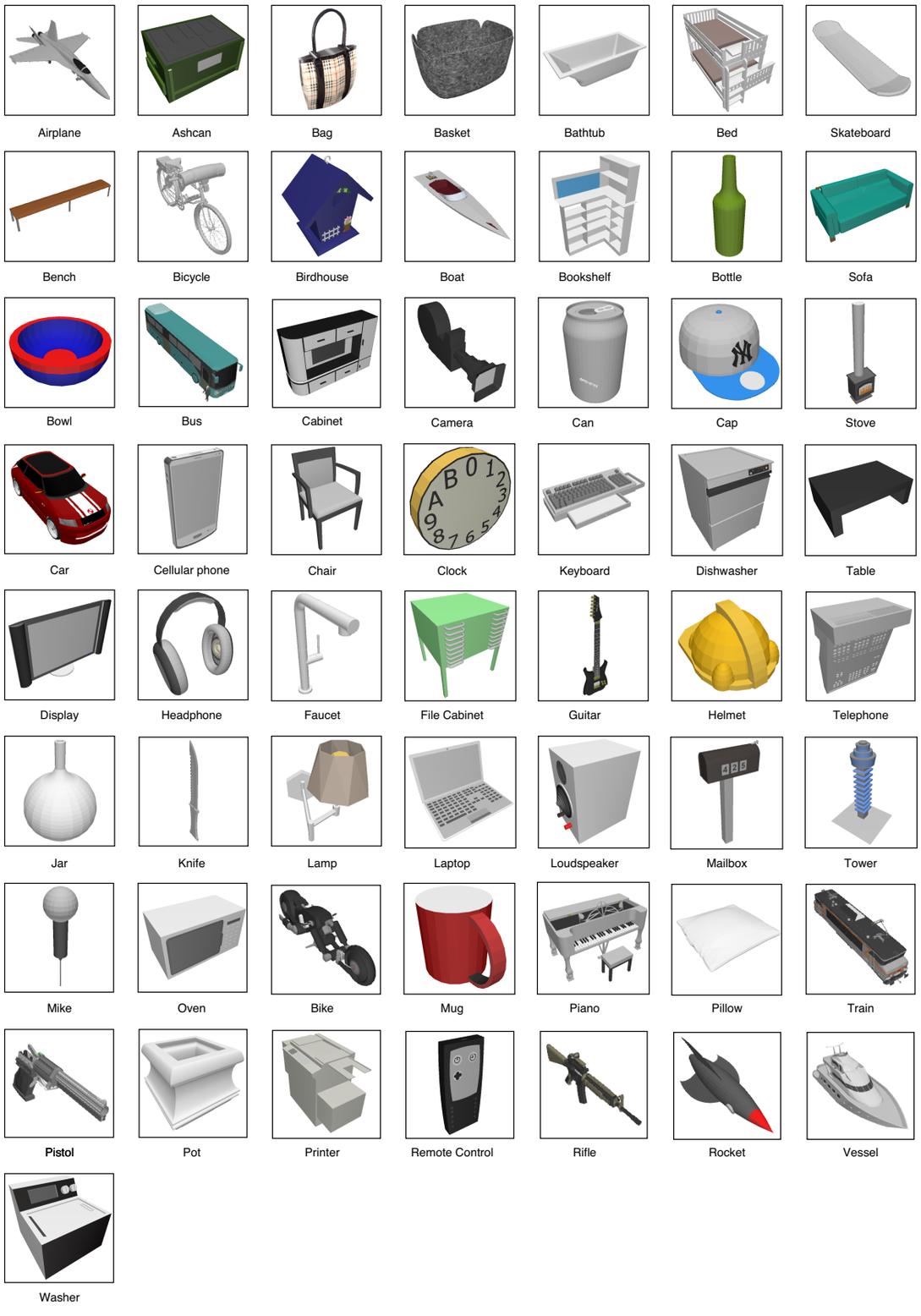


図 21　**ShapeNetCore データセット:** ShapeNetCore には 57 種類のカテゴリーがある．図にはそれぞれのカテゴリーからひとつずつ取ってきた 3D モデルを示した．

5.1 Quantitative Evaluation

■ 閾値 ニューラルネットが出力するデータの値は 0 から 1 の実数であるため, 最後は閾値を境目にボクセルの領域と空の領域に分割する. しかし, この二値化処理を行うと空中に浮いてしまうボクセルが出てくることがある (図 22). このようなボクセルは閾値を調整することによって減る可能性がある. そこで, 閾値を変えることで浮いてしまうボクセルを減らすことができないか調べた. 具体的には生成されたボクセルモデルに対して, 独立している (三次元グリッド上において連続的につながっていない) 領域の数を調べる. 例えば, 図 22 の場合, 領域数は 7 である. 領域を数えるときは, すべてのボクセル領域が塗りつぶされるまで flood fill を繰り返し, flood fill を行った回数をカウントする. また, 前処理として標本にはボクセル領域の膨張, 収縮 (region growing) を行っておく. 提案手法によって生成された 50,000 の標本に対して, 0.1 から 1.0 まで 0.1 ステップずつ閾値を変えて領域数を調べた結果を図 23 に示す. この図を見ると, 閾値によって領域数の平均値が大きく変わることはないが, 最も平均値が小さくなるのは閾値 0.1 の 2.59 であることが分かった. また, 閾値が大きくなるにつれ平均値も少しずつ上がっていくことが分かった. 次に閾値 0.1 のときの領域数を度数分布で表したところ, およそ半分のデータは領域数が 1 であった. 統計的に見れば, 次にボクセルモデルを生成したときに領域数が 1 である確率は約 0.54, 領域数が 2 以上である確率は約 0.46 であるといえる. つまり, 今回の実験で空中に浮いてしまうボクセルが出てくる確率はおよそ 1/2 である.

■ 計算速度 提案手法がひとつのボクセルモデルを生成するのにかかる時間を計測した結果を表 1 に示す. ここでは評価基準として, 自己符号化器型ニューラルネット (AE-like net) を用いる. 自己符号化器型ニューラルネットも同じデータセットを用いて約 12 時間訓練した. 評価基準としたネットワークモデルの詳細は https://github.com/maxorange/pix2vox から見ることができる. 処理時間は単一の物理 CPU または GPU で計算した場合をそれぞれ計測した. CPU は *Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz* を, GPU は *NVIDIA GeForce GTX TITAN X* を用いた. 表 1 に示されている処理時間はひとつのボクセルモデルを 100 回生成した時間の平均である. CPU 上では提案手法は 10.607 秒となり, AE-like net の方が約 4 秒速かった. しかし, GPU 上ではその差はなくなり, 処理時間は CPU のときの約 1/560 に縮まった. このことから, GPU 上であればほぼリアルタイムでボクセルモデルを生成できるといえる.

Method	CPU (sec)	GPU (sec)
AE-like net	6.138	0.019
提案手法 (proposed)	10.607	0.019

表 1 **計算速度**

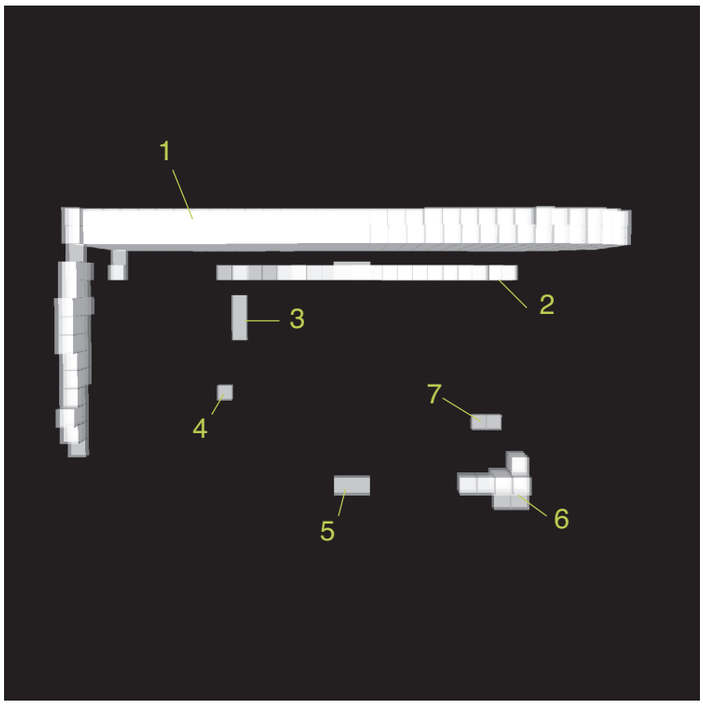
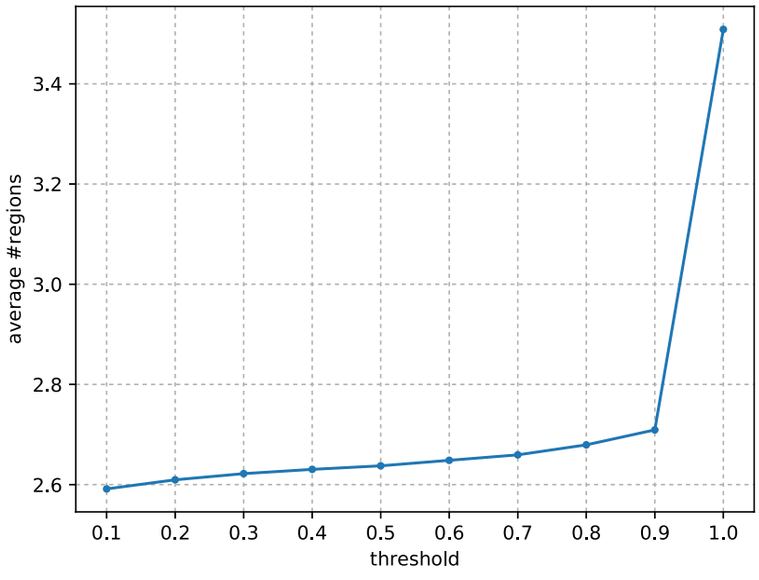


図 22　**空中に浮いてしまうボクセル:** この場合, 領域数は 7.

Threshold	Average
0.1	2.59
0.2	2.60
0.3	2.62
0.4	2.63
0.5	2.63
0.6	2.64
0.7	2.65
0.8	2.67
0.9	2.70
1.0	3.50

(a)

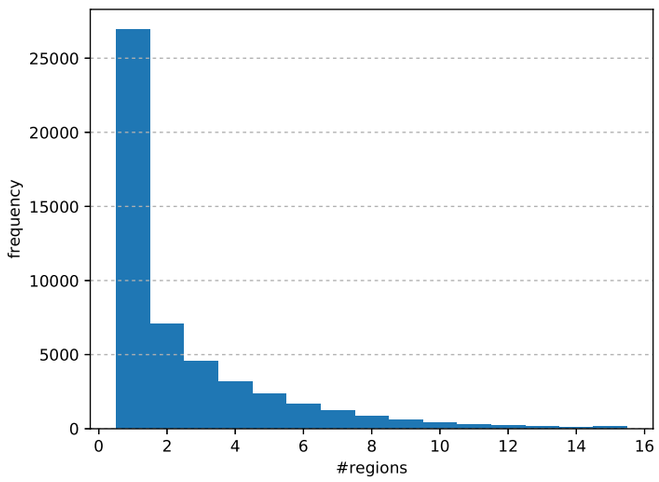


(b)

図 23　**閾値を変えて領域数を調べた結果:** 閾値ごとの領域数の平均値を (a) に, グラフに表したものを (b) に示した.

#regions	Frequency	Percent
1	26,947	53.9
2	7,062	14.2
3	4,583	9.2
4	3,194	6.4
5	2,383	4.8
6	1,692	3.4
7	1,238	2.5
8	847	1.6
9	593	1.1
10 -	1,461	2.9
Total	50,000	100.0

(a)



(b)

図 24　**閾値 0.1 としたときの領域数の度数分布:** (a) と (b) は同じデータをそれぞれ表とヒストグラムにしたものである.

5.2　Qualitative Evaluation

実際に人が描かいたスケッチからボクセルモデルを生成した結果を図 25 に示す. 図中では左に入力スケッチ, その右隣にボクセルモデルを配置している. 入力スケッチの灰色の破線は線描として第 1 ステップの生成器に入力される. また, 実線は色情報として第 2 ステップの生成器に入力される. 図 25 では評価基準として, 5.1 節で用いた自己符号化器型ニューラルネット (AE-like net) から生成したボクセルモデルも並べてある. ボクセルの領域と空の領域に分割するときの閾値はすべて 0.1 とした. 結果を見てみると, AE-like net では入力されたスケッチの色が淡くなってしまっているのに対して, 提案手法でははっきりと色が付いている. さらに, 提案手法は色の位置もある程度正確に生成できることが分かった. ボクセルモデルを生成するときのデモ動画は https://drive.google.com/file/d/0B4HwtOC5zptJWkRlTUZvcW9uZW8/view?usp=sharing から見ることができる. ShapeNet のデータセットを使って分かったことは, 椅子の脚や飛行機の羽といった細長い形はあまりうまく生成されなかったということである. また, 訓練データは CAD モデルであるため角張った形が多いが, 実際はそれよりも丸みを帯びた形が生成されている. 提案手法は CAD モデルより動物や植物といった有機的な形の生成に向いていると考えられる.

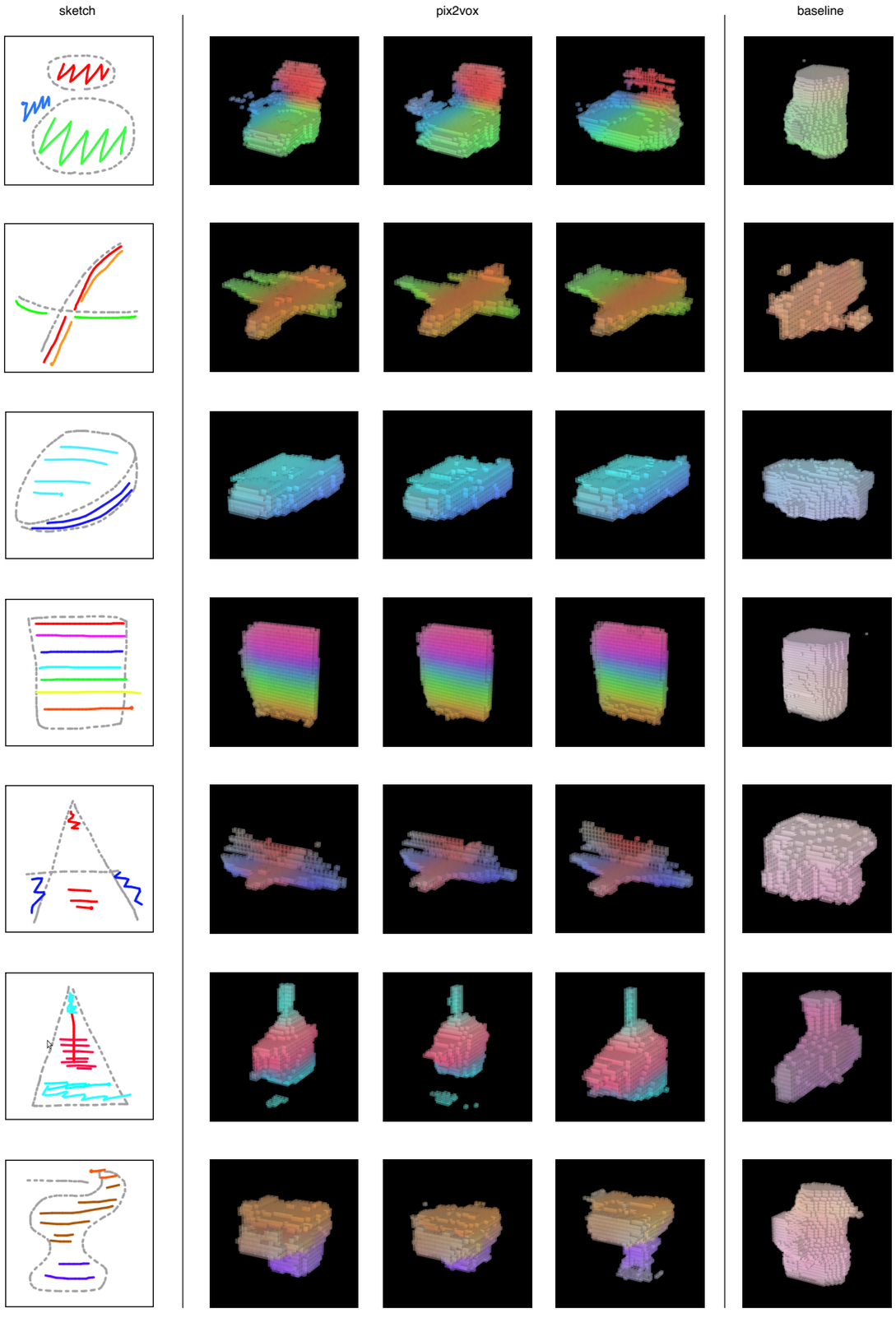


図 25　**定性的評価:** 図中のそれぞれの行は, 入力したスケッチ, 提案手法で生成したボクセルモデル, そして, 評価基準とする自己符号化器型ニューラルネットで生成したボクセルモデルという順で並んでいる. 詳細は 5.2 節を参照.

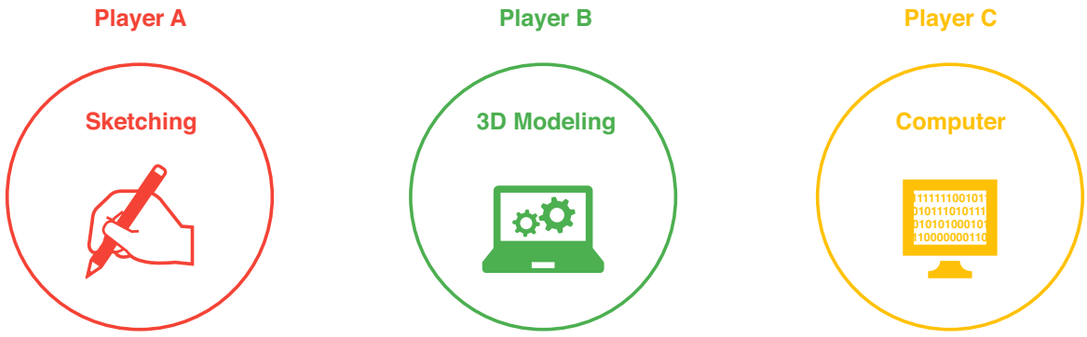


図 26　**3 種類のプレイヤー:** 本研究で行ったユーザ実験には 3 種類のプレイヤーがいる．絵を描く Player A. 3D モデリングを行う Player B. 最後は Computer である．ここで Computer とは本研究で提案する手法を指す．詳しくは 5.3 節を参照.

5.3　User Experiments

本研究の特徴はひとつのスケッチから複数のボクセルモデルを提示できることである．しかし, あるスケッチに対してどのようなボクセルモデルを出力すれば良しとするのか客観的評価が困難な点がある．その上, 他の手法と性能を比較することも難しい．このような場合, ひとつの指標となり得るのは生成した 3D モデルの質を人間に評価してもらうことである．そこで, 4 節の手法で生成した 3D モデルと人間が作った 3D モデルを並べ, どれが人間の 3D モデルかを選んでもらう実験を行った．実験の目的はどこまで人間の作ったものと判別がつかない 3D モデルを生成できるかを見ることである．実験参加者は田中浩也研究室のモデリングソフトに使い慣れている学生である．便宜上, 本節では 4 節の手法を Computer と呼ぶことにする.

■ Details　この実験には 3 種類のプレイヤーが存在し (図 26), 次の 4 つのステップから成る.

- **Step 1:** 実験参加者はペアになる．一方は絵を描く人 (以下 Player A) になり, 他方は 3D モデリングをする人 (以下 Player B) になる.
- **Step 2:** Player A は与えられたお題にしたがって絵を描く．例えば, ウサギを描いてくださいと言われたらウサギを描く.
- **Step 3:** Player A によって描かれた絵をもとに, Player B は 3D モデルを一つ作る, Computer は 3D モデルを複数生成する.
- **Step 4:** 3D モデルを Player A に見せ, Player B が作ったものを選んでもらう．正しく選択できた場合は, Computer は判別のつかない 3D モデルの生成に失敗したとする．誤って Computer の 3D モデルを選択した場合は, 判別のつかない 3D モデルの生成に成功したとする.

以上 4 つのステップを図 28 に示した．整理すると Player A は絵を描く人, Player B はモデリングをする人である．次に Player A と Player B が行う作業の詳細を述べる．

Player A には図 27 の Web インターフェースを通じて絵を描いてもらう (次の URL から見ることができる: https://maxorange.github.io/sketch/)．Web インターフェースの機能は単純であり, 線を描く機能と消す機能がある．これらの機能は JavaScript の Canvas を使って開発した．キャンバスの大きさは 500 × 500 ピクセル, 線の太さは 10 ピクセルにした．線の太さと色は一定で変えられない．Player A が絵を描く時間は 1 分に設定した．

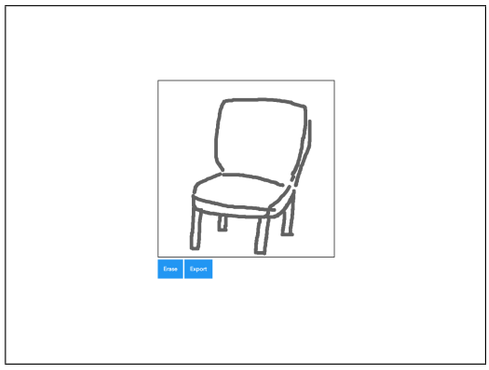


図 27　**Player A が絵を描くための Web インターフェース:** Player A にはこの上で絵を描いてもらう．図中の絵はデモとして描いたものである．

Player B には既存の CAD で 3D モデルを作ってもらう．どの CAD を使うかは自由に選んでよいことにした．理由は CAD を指定すると, その CAD の特性から人間が作ったものがすぐに分かってしまうからである．もちろん, Player A はどの CAD を使ったのか知らない．また, Player B がモデリングしたデータは Computer が生成したボクセルモデルと合わせるために, 32 × 32 × 32 の大きさでボクセル化を行った．今回の実験では Player B と Computer の両者ともボクセルの色はなしとした．Player B がモデリングを行う時間は 3 分に設定した．

■ Results　実際に「車」と「飛行機」をお題にして実験を行った結果をそれぞれ図 29, 30 に示した．実験時においては両方とも人間によって作られた 3D モデルではなく Computer によって生成されたボクセルモデルが選ばれた．上述の Step 4 に従えば Computer は判別できない 3D モデルの生成に成功したといえる．しかし, よく見れば判別できないことはない．実験後, 参加者以外の人に結果をみせたところ, 「車」は人間が作った 3D モデルが選ばれた．この実験結果は https://htanakalab.github.io/tmoriya/thesis/bachelor/user-expt/ から 3D ビューで見ることができる．答えは付録 B に記してある．あなたは人間が作った 3D モデルを当てることができるだろうか．

図 28　**ユーザ実験の手順:** 実験の大まかな流れは以下のようになっている．Step 1) 実験参加者は Player A と Player B に分かれる．Step 2) PlayerA は与えられたお題 (例えば「動物」) に従って絵を書く．Step 3) Player A によって描かれた絵をもとに, Player B と Computer は 3D モデルを作る．Player B が作る 3D モデルは 1 つであるのに対して, Computer は複数の 3D モデルを作る．Step 4) 3D モデルを Player A に見せ Player B が作った 3D モデルを選んでもらう．

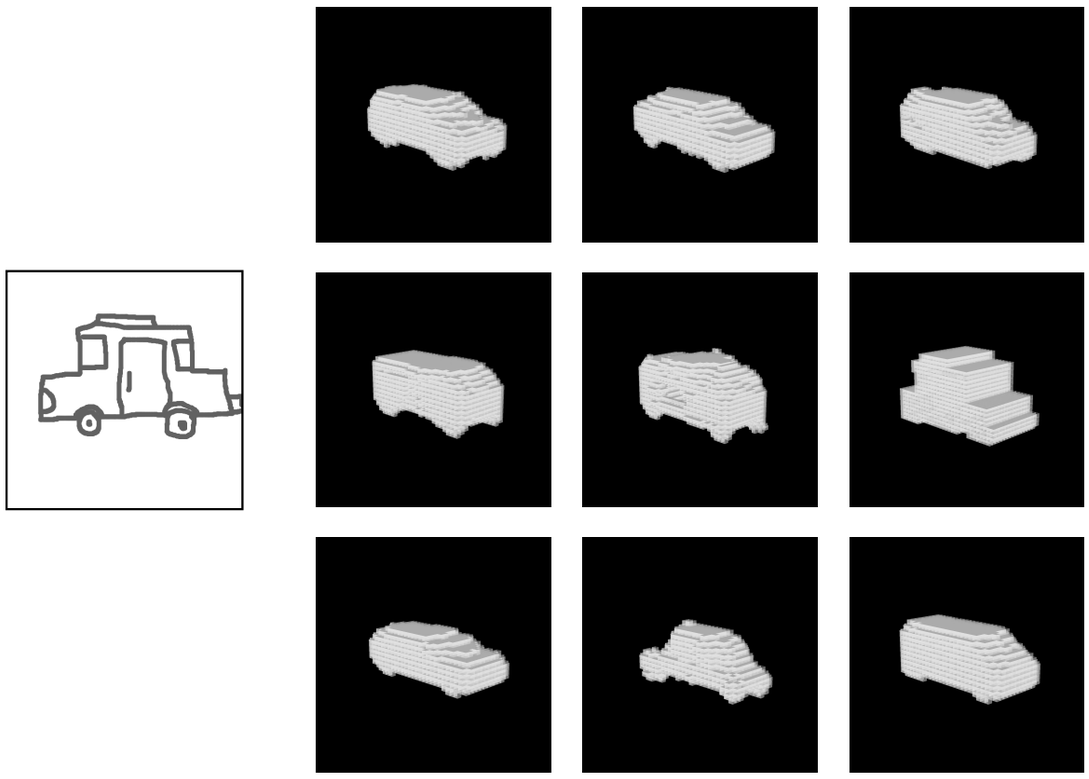


図 29　**実験 1:** お題は「車」にして実験を行った結果. この中の一つは人間が作ったモデルで, その他のモデルは Computer によって生成されたものである. 詳細は 5.3 節を参照.

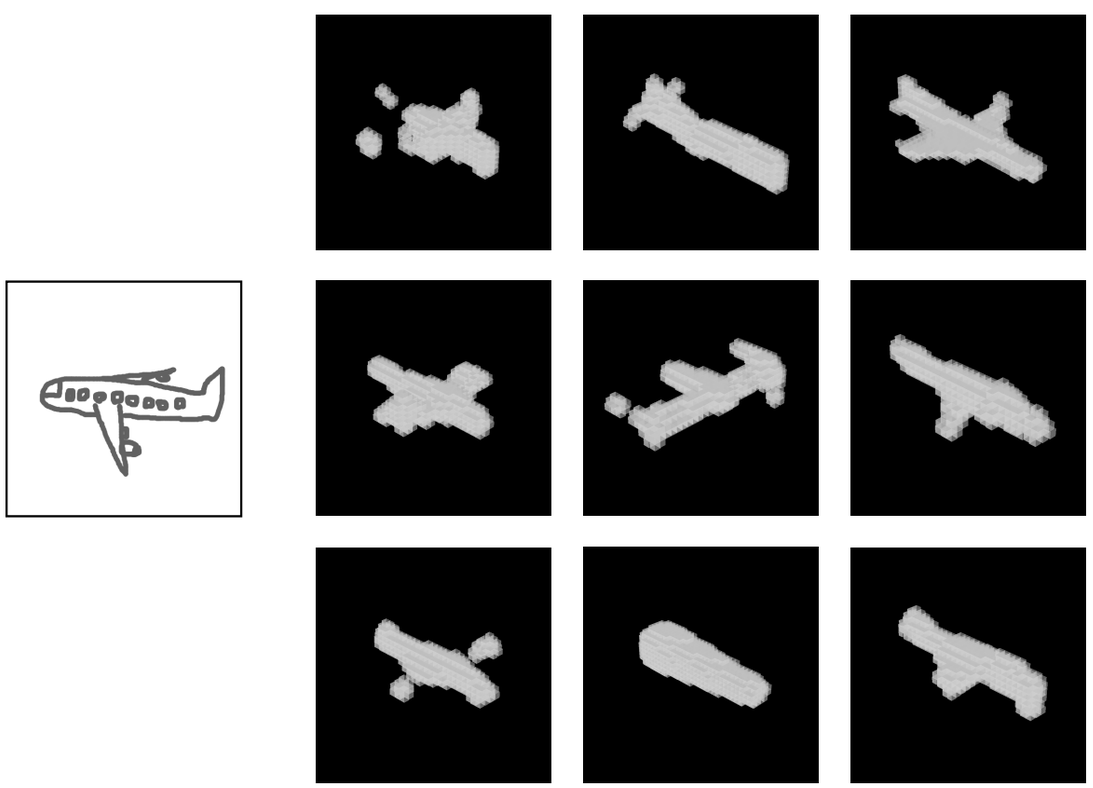


図 30　**実験 2:** お題は「飛行機」にして実験を行った結果. 「車」の実験と同様にこの中の一つは人間が作ったモデルである.

6 Discussion

6.1 Artificial Intelligence vs. Machine Learning

現時点では人工知能, AI ではなく, 機械学習と呼ぶのがよいのではないだろうか. というのも, メディアでは, 人工知能が仕事を奪うなど, いかにもすごい技術のように取り上げられているが, 実際はそこまでではない. 少なくとも現時点では鬼気迫るものではない. もしパターンで解くのではなく, 言葉や問題の意味を理解して解くようになったらどうなるかわからない. 中身を知ってしまうと果たしてこれが知能と呼べるものなのだろうかという疑問が出てくる. 人工知能と呼ばれているものが中でやっていることは計算だからである. 2016 年に話題となった AlphaGo [37] も基礎は強化学習という単純なアルゴリズムが使われている. 革新的なまったく新しい技術があるのではなく, 既存の技術を組み合わせたものでできている. もしかしたら私たちも複雑系のように単純なアルゴリズムの組み合わせでできているかもしれないが. 脳の研究は現在盛んに行われている. しかし, まだ脳の全貌は明らかになっていない. 知能自体の全貌が明らかになっていないのにも関わらず人工知能と呼ぶのは少し早いのではないだろうか.

では, 一体知能とは何であろうか.

6.2 Digital Fabrication

本研究では既存の CAD によるモデリングが 3D プリントを行うまでの壁となっているという問題意識から開発を進めてきた. しかし, 私は誰でも使えるようなモデリングソフトを作りたいだけではない. 人間が二次元の映像からどのようにして三次元の物体を認識しているのかにも興味がある. ここで述べている三次元物体の認識は両眼による立体視ではない. 私がここで指しているのは, 例えば, カメラで撮影された写真を見てもその中に写っている物体の三次元形状を推測することができる能力のことである. 二次元の映像からどのようにして三次元の物体を認識しているのかという問いをさらに突き詰めていくと, 最終的にどのようにして意識は生まれるのかという問いにたどり着く. 意識は一体どうなっているのだろうと疑問に思ったことはないだろうか. すると突然あたりまえのように過ごしてきた日常が不思議に思えてこないだろうか.

ところでデジタルファブリケーションとは情報世界のモノを物理世界のモノに出力することと, その逆を繰り返すことである. 別の言い方をすればモノが情報世界と物理世界を行き来することである. そうだとしたら人がモノを作ることそのものもデジタルファブリケーションということができるのではないか. 例えば, 粘土をこねて何かを作った経験はないだろうか. 粘土で何かを作るというと様々

なものが思いつくが, ここでは頭の中で犬を作りたいと思っていたとする. そうしたらまずは粘土をこね, 犬の形を作ってみる. 今度は作ってみた形を見て (フィードバックを得て), だんだんと自分の理想とする形に変えていく. このようにして人は「作る」「見る」という行為を繰り返してモノを作っていく. 私たちの意識は情報であるという解釈をすれば, 「作る」は脳の中の情報を物理的な形にする行為であり, 「見る」は作った形をまた脳の中に戻す行為である. このことから人がモノを作る行為もデジタルファブリケーションということができるのではないだろうか.

神経細胞は他の神経細胞からの信号を受けて発火したり抑制したりする. 発火するときを 1, 抑制するときを 0 とすれば, 神経回路における情報処理にはコンピュータと同じように 2 値の信号が使われているとも考えることができる. つまり, 私たちの神経系は 2 値の信号によって情報化された世界である. 外界からの刺激が目や耳などの感覚器官によって電気信号に変換され, それが脳に伝達されていく. 全く同じというわけではないが, コンピュータと脳には類似している部分がある. 例えば, コンピュータにおけるデジタルデータは CAD でモデリングされた 3D モデルであり, 人におけるデジタルデータは頭の中にあるぼんやりとした形である. 頭の中にある形も神経細胞の活動 (発火・抑制) によって生まれている. こういったことから人がモノを作る行為もデジタルファブリケーションではないかということを上で述べた. では, コンピュータがモノを作るのと人がモノを作るのとでは何が違うのか.

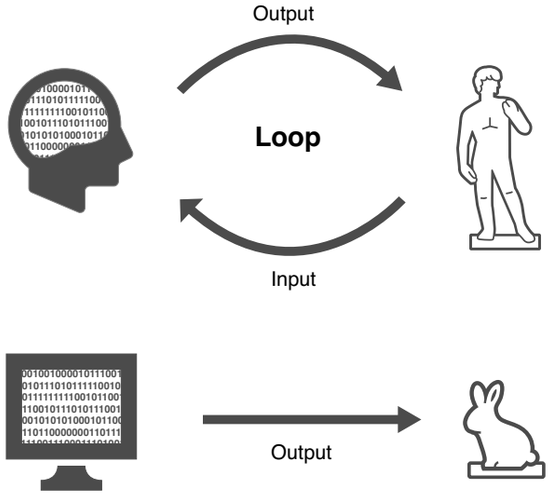


図 31　**人とコンピュータがモノを作るときの違い.**

人とコンピュータがモノを作るときの違いは入力と出力を繰り返すことができるかできないかである (図 31). 人は作る (出力) と見る (入力) を繰り返すことでモノを作る. 一方でコンピュータは STL や OBJ といったフォーマットに従って作る前からモノの形が明確に記述されており, 後はそれを 3D プリンタによって出力するだけである. 確かに 3D プリンタで出力したモノを人が 3D スキャンしてまた 3D データに戻せば, コンピュータでも入力と出力を繰り返せるではないかという考え方もある.

本来, 冒頭で定義したデジタルファブリケーションは 3D プリンタや 3D スキャナによってモノが情報世界と物理世界を行き来することを指している. しかし, そうすると人とコンピュータの境界線が曖昧になるため, 両者の違いという点では考慮に入れないことにする.

両者の違いは他にもある. 例えば, コンピュータでは作る前からモノの形が明確に記述されていると述べたが, 人が何かを作るときは最初から自分の作りたいものがはっきりと頭の中にあるわけではない. 最初は頭の中にぼんやりと自分の作りたいものがあり, 手を動かし実際に出力してみることによってだんだんと具体的な形が見えてくるのである. このようにして入力と出力を繰り返すことで人の創造性は生まれるのではないだろうか. アートやデザインの世界で作品に完成がないと言われているものはまさに「作る」「見る」というループの罠にはまっている状態である. しかし, 心配しすぎる必要はない. なぜなら, ときに締め切りがこのループを打破してくれるからである.

いままでは人とコンピュータを分けて考えてきたが, 両者は密接に関わり合っている. 実際, コンピュータ上の 3D データを作っているのは人であり, 3D プリンタでそのデータを出力させているのも人である. このようにして人とコンピュータとモノの間には図 32 のような関係がある. 図 32 中のコンピュータとモノをつなぐ有向線 (デジタルファブリケーション) は三者の関係があって初めて成り立つ. デジタルファブリケーションはもともと人に備わっていたものであり, コンピュータや工作機械がそれを拡張したとも考えることができるのではないだろうか.

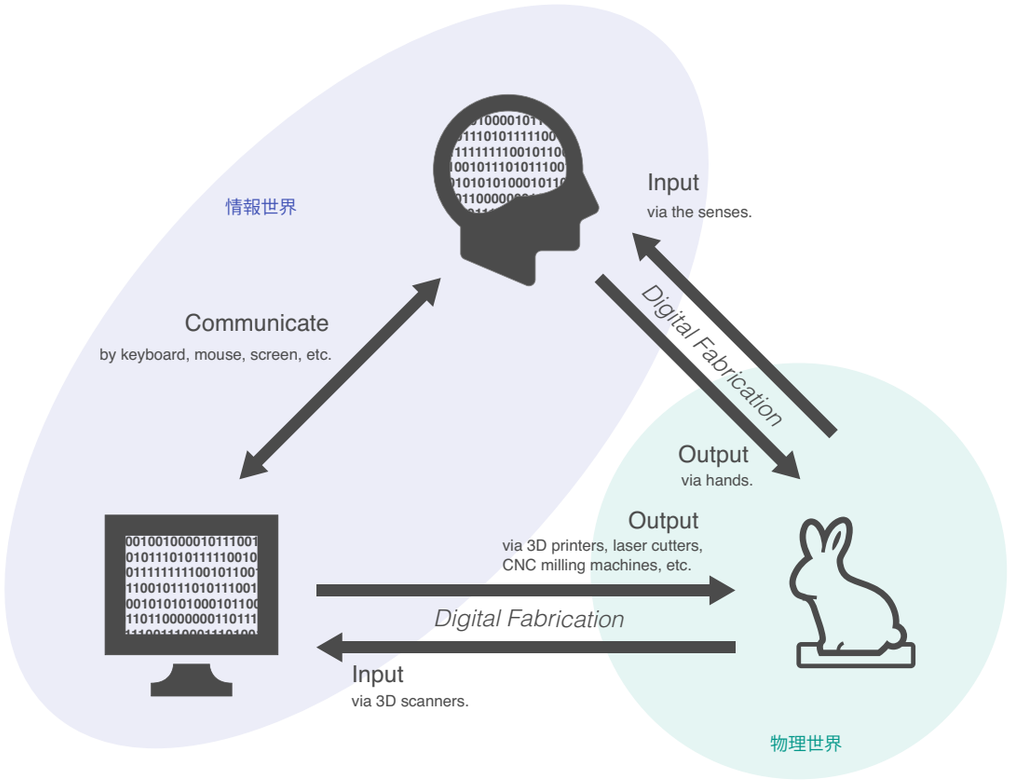


図 32　**ものづくりにおける人とコンピュータとモノの関係.**

■ 最後に　デジタルファブリケーションと同じように, 私が本節で述べてきたことも, 頭の中で行われた神経活動を言葉として出力したものである. しかし, 考えているだけではきりがないため, ここで議論は止める. 人はどんなものにも理由や意味を付けたがるくせがあるが, 本当は特に理由や意味はないのかもしれない. 私たちが存在していることも, 宇宙が存在していることも特に理由はないのかもしれない. 理由や意味を付けたがるくせがあったからこそ, 人は高度な文明を築き上げることができたのかもしれないが.

7　Conclusion and Future Work

以上, 本研究では Generative Adversarial Networks を通じたスケッチによる 3D モデリングシステムの開発を行った. 4 節では GAN を用いてスケッチから着色されたボクセルモデルを生成する手法について述べ, その手法をもとにリアルタイムでスケッチからボクセルモデルを生成するインターフェースを実装した. 5 節では定性的・定量的評価の二つとユーザ実験を行った. ユーザ実験とは生成した 3D モデルと人間が作った 3D モデルを並べ, 人間の作ったものを当ててもらうテストであり, 結果は五分五分となった. 今回の実験では ShapeNet の CAD データを $32 \times 32 \times 32$ の解像度にボクセル化して使用したが, 今後はより解像度の高いボクセルモデルを扱えるようにし, 3D データのバリエーションも増やしていきたい. また, 本研究の特徴はスケッチを入力する度に毎回異なる形がでてくることである. そのため単なるモデリングツールとしての役割を果たすだけでなく, 人が新しいアイデアを考えるときの支援にもなる可能性があるのではないだろうか. 最後に GAN によって生成したボクセルモデルを 3D プリントしたものを図 33 に示す.

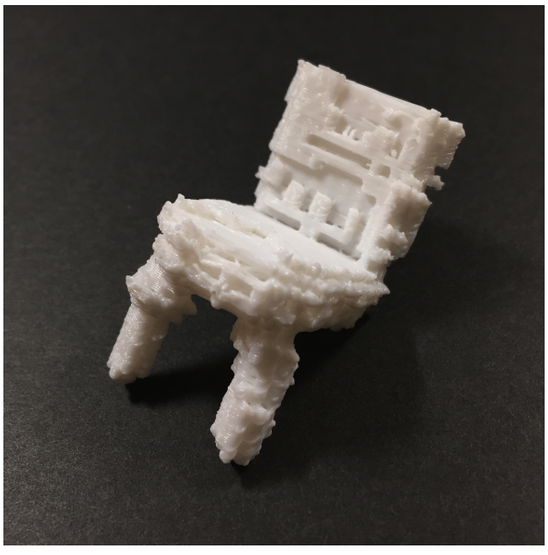


図 33　**3D プリント:** これは図 17 で作成した前脚のみの椅子を 3D プリントしたものである. もし椅子を作ってくださいと言われたら, 私たちは前脚のみの椅子を作るだろうか.

Acknowledgements

本研究を進めるにあたり, 研究のまとめ方や評価方法だけでなく物事の考え方から研究に対する姿勢まで, 3 年半にわたってさまざまな局面でご指導をいただいた, 田中浩也教授に心より感謝いたします. 田中浩也研究室の OB である升森敦士氏には本研究の初期の頃から, 特に提案手法に関して, たくさんの的確なアドバイスをいただきました. ここに感謝の意を表します. 5.3 節での実験に際して快く協力していただき, さらに実験に対する有益なフィードバックをくださった, 田中浩也研究室の先輩・同期・後輩の皆様に感謝いたします. また, 同研究室の先輩・同期の皆様には研究の指針となるたくさんの意見もいただきました. ここに感謝の意を表します.

Appendix

A Hyperparameters

本研究で用いたネットワーク構造を表 2, 3, 4, 5, 6, 7 にまとめる．これらの表中では上から入力層, 隠れ層, 出力層の順に並んでいる．すべてのネットワークにおいて入力層と出力層以外に Dropout [39] を適用する．ノードを消す割合は一律に 25% とする．また, 隠れ層の活性化関数には ELU [8] を, 最適化には Adam [24] を用いる．[31] にならって GAN の訓練時は $\beta_1 = 0.5$ とする．学習率はそれぞれの表題に記す．表 2, 3 の符号化ネットワークは事前に訓練し, 最後の全結合層は GAN の訓練時には用いない．最後から二番目の層の出力を GAN の訓練時に用いる．

Operation	Kernel	Strides	Feature maps	Activation
Convolution	4×4	2×2	32	ELU
Convolution	4×4	2×2	64	ELU
Convolution	4×4	2×2	128	ELU
Convolution	4×4	2×2	256	ELU
Linear	-	-	57	Softmax

表 2　**Encoder (sketch):** 事前訓練時の学習率は 0.0005．ミニバッチサイズは 64.

Operation	Kernel	Strides	Feature maps	Activation
Convolution	4×4	2×2	64	ELU
Convolution	4×4	2×2	128	ELU
Convolution	4×4	2×2	256	ELU
Convolution	4×4	2×2	512	ELU
Linear	-	-	57	Softmax

表 3　**Encoder (RGB image):** 事前訓練時の学習率は 0.0005．ミニバッチサイズは 64.

Operation	Kernel	Strides	Feature maps	Activation
Linear (noise and class label)	-	-	$4 \times 4 \times 4 \times 128$	ELU
Convolution	$4 \times 4 \times 4$	$1 \times 1 \times 1$	128	ELU
Transposed Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	64	ELU
Transposed Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	32	ELU
Transposed Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	16	ELU
Transposed Convolution	$3 \times 3 \times 3$	$1 \times 1 \times 1$	1	Sigmoid

表 4　**Voxel Generator:** スケッチは表の上から二番目の層で与える．スケッチはノイズ, クラスラベルの特徴量とチャンネル方向に沿って連結する．学習率は 0.0004．ミニバッチサイズは 64.

Operation	Kernel	Strides	Feature maps	Activation
Convolution	$4 \times 4 \times 4$	$1 \times 1 \times 1$	16	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	32	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	64	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	128	ELU
Transposed Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	128	ELU
Convolution	$4 \times 4 \times 4$	$1 \times 1 \times 1$	128	ELU
Transposed Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	64	ELU
Convolution	$4 \times 4 \times 4$	$1 \times 1 \times 1$	64	ELU
Transposed Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	32	ELU
Convolution	$4 \times 4 \times 4$	$1 \times 1 \times 1$	3	Tanh

表 5　**Color Generator:** RGB 画像は上から五番目の層で与える．画像はボクセルモデルの特徴量とチャンネル方向に沿って連結する．学習率は 0.0002．ミニバッチサイズは 32.

Operation	Kernel	Strides	Feature maps	Activation
Convolution	$4 \times 4 \times 4$	$1 \times 1 \times 1$	16	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	32	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	64	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	128	ELU
Linear	-	-	2	Softmax
Linear (class label)	-	-	57	Softmax
Linear (noise)	-	-	100	Tanh

表 6　**Voxel Discriminator:** 学習率は 0.0001．ミニバッチサイズは 64．このネットワークには 3 種類の出力がある．

Operation	Kernel	Strides	Feature maps	Activation
Convolution	$4 \times 4 \times 4$	$1 \times 1 \times 1$	32	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	64	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	128	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	256	ELU
Convolution	$4 \times 4 \times 4$	$2 \times 2 \times 2$	512	ELU
Linear	-	-	2	Softmax

表 7　**Color Discriminator:** 学習率は 0.0001．ミニバッチサイズは 32．最後から二番目の層の出力には [35] の minibatch discrimination を適用し, 次の層に入力する．

B Answer

5.3 節の実験結果の答えを表 8 に記す.

お題	答え
車	2 行目の 3 列目
飛行機	3 行目の 2 列目

表 8 実験結果の答え

References

[1] Pyqt. https://riverbankcomputing.com/software/pyqt/intro. Accessed: 2016-12-15. 23

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org. 23

[3] Blender. https://www.blender.org. Accessed: 2016-12-15. 7

[4] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986. 25

[5] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. Shapenet: An information-rich 3d model repository. *arXiv*, 2015. 25

[6] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*. 2016. 8

[7] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *ECCV*, 2016. 8

[8] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*, 2016. 41

[9] E. L. Denton, S. Chintala, a. szlam, and R. Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*. 2015. 8

[10] M. W. Diederik P. Kingma. Auto-encoding variational bayes. In *ICLR*, 2014. 8

[11] M. Eitz, R. Richter, T. Boubekeur, K. Hildebrand, and M. Alexa. Sketch-based shape retrieval. *SIGGRAPH*, 31(4):31:1–31:10, 2012. 8

[12] Ephtracy. Magicavoxel. https://ephtracy.github.io. Accessed: 2016-12-15. 5

[13] J. Gauthier. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N*, 2014. 8

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*. 2014. 8, 16

[15] F. Han and S.-C. Zhu. Bayesian reconstruction of 3d shapes and scenes from a single image. In *Proceedings of the First IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis*, 2003. 8

[16] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006. 8

[17] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002. 8

[18] Q. Huang, H. Wang, and V. Koltun. Single-view reconstruction via joint analysis of image and shape collections. *SIGGRAPH*, 34(4):87:1–87:10, 2015. 8

[19] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie. Stacked generative adversarial networks. *arXiv:1612.04357*, 2016. 8, 21

[20] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3d freeform design. In

SIGGRAPH, 1999. 4

[21] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv:1611.07004*, 2016. 8

[22] H. Izadinia, Q. Shan, and S. M. Seitz. Im2cad. *arXiv:1608.05137*, 2016. 8

[23] A. Kar, S. Tulsiani, J. o Carreira, and J. Malik. Category-specific object reconstruction from a single image. In *CVPR*, 2015. 8

[24] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 25, 41

[25] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. 10

[26] Meshmixer. http://www.meshmixer.com. Accessed: 2016-12-15. 7

[27] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014. 8

[28] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv:1609.03126*, 2017. 8, 21

[29] OpenSCAD. http://www.openscad.org. Accessed: 2016-12-15. 6

[30] Qubicle Voxel Editor. http://www.minddesk.com. Accessed: 2016-12-15. 5

[31] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 8, 25, 41

[32] S. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee. Learning what and where to draw. In *NIPS*. 2016. 8

[33] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In *ICML*, 2016. 8

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. 16

[35] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NIPS*. 2016. 8, 42

[36] Sculptris. http://pixologic.com/sculptris/. Accessed: 2016-12-15. 7

[37] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. 35

[38] J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *ICLR*, 2016. 8

[39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014. 41

[40] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3d models from single images with a convolutional network. *arXiv:1511.06702*, 2016. 8

[41] Thingiverse. https://www.thingiverse.com. Accessed: 2016-12-15. 6

[42] Vectary. https://www.vectary.com. Accessed: 2016-12-15. 6

[43] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *NIPS*. 2016. 8

[44] VoxelShop. https://blackflux.com. Accessed: 2016-12-15. 5

[45] F. Wang, L. Kang, and Y. Li. Sketch-based 3d shape retrieval using convolutional neural networks.

In *CVPR*, 2015. 8

[46] X. Wang and A. Gupta. Generative image modeling using style and structure adversarial networks. In *ECCV*, 2016. 8

[47] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a probabilistic latent space of objectshapes via 3d generative-adversarial modeling. In *NIPS*. 2016. 8

[48] Q. Yu, Y. Yang, Y. Song, T. Xiang, and T. M. Hospedales. Sketch-a-net that beats humans. In *BMVC*, pages 7.1–7.12, 2015. 8

[49] Q. Yu, F. Liu, Y.-Z. SonG, T. Xiang, T. Hospedales, and C. C. Loy. Sketch me that shoe. In *CVPR*, 2016. 8

[50] K. Yucer, C. Kim, A. Sorkine-Hornung, and O. Sorkine-Hornung. Depth from gradients in dense light fields for object reconstruction. In *3DV*, pages 249–257, 2016. 8

[51] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. Sketch: An interface for sketching 3d scenes. In *SIGGRAPH*, pages 163–170, 1996. 4

[52] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv:1612.03242*, 2016. 8

[53] J. J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *arXiv:1609.03126*, 2017. 8

[54] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros. Generative visual manipulation on the natural image manifold. In *ECCV*, 2016. 8, 22