
基礎演習テキスト

椋山女学園大学現代マネジメント学部 三木 邦弘
令和7年9月23日版

| | | | | | |
|------|---------------------------|----|------|------------------------------|----|
| 1 | はじめに | 2 | 4.1 | Blender の起動方法 | 19 |
| 2 | Linux (コンピュータ管理ソフト) | 4 | 4.2 | シーンに対する操作 | 20 |
| 2.1 | ユーザー登録 | 4 | 4.3 | オブジェクトに対する操作 | 21 |
| 2.2 | リモートデスクトップによる接続 | 5 | 4.4 | オブジェクトの変形などの操作 | 24 |
| 2.3 | パスワードの変更 | 7 | 4.5 | オブジェクトなどの保存 | 26 |
| 2.4 | インストール済みのアプリ | 7 | 4.6 | プレートの作成 | 26 |
| 3 | 文書清書システム | 8 | 5 | 3D プリンター | 28 |
| 3.1 | Texmaker の起動 | 8 | 5.1 | 3D プリンターの使い方 | 28 |
| 3.2 | 簡単な例 | 8 | 5.2 | 3D プリンター使用上のヒント | 31 |
| 3.3 | 印刷する方法 | 9 | 6 | 拡張現実 | 32 |
| 3.4 | LaTeX のコマンド | 9 | 6.1 | AR テストシステムの使い方 | 32 |
| 3.5 | 使用できる文字 | 10 | 6.2 | ブラウザによる AR システム | 34 |
| 3.6 | 文章のスタイル | 10 | 6.3 | テキストチャ付きモデルの作成方法 | 36 |
| 3.7 | タイトルページ | 10 | 6.4 | マーカー例 | 37 |
| 3.8 | 文章構成 | 11 | 7 | ChatGPT | 39 |
| 3.9 | 空白 | 11 | 7.1 | ゲームプログラムを作る | 40 |
| 3.10 | 文字のスタイル | 11 | 7.2 | ChatGPT をどのように捉えるか | 49 |
| 3.11 | 行、段落 | 12 | 8 | 深層学習 | 51 |
| 3.12 | 行の空け方 | 13 | 8.1 | コンピュータによる問題解決 | 51 |
| 3.13 | 中揃え・右揃え | 13 | 8.2 | 機械学習 | 51 |
| 3.14 | 箇条書 | 13 | 8.3 | ニューラルネットワーク | 52 |
| 3.15 | そのままの形 | 14 | 8.4 | 様々なフレームワーク | 54 |
| 3.16 | 表 | 14 | 8.5 | 画像認識プログラムの例 | 54 |
| 3.17 | 脚注の入れ方 | 15 | 8.6 | 画像認識プログラムの実行 | 57 |
| 3.18 | 数式 | 15 | 8.7 | 学習プログラムの例 | 58 |
| 3.19 | 画像の入れ方 | 16 | 8.8 | 学習プログラムの実行 | 61 |
| 3.20 | 図表の参照 | 16 | 8.9 | 学習用データを増やす | 63 |
| 3.21 | 参考文献リスト | 17 | 8.10 | 水増し学習プログラムの実行 | 64 |
| 4 | 三次元モデルの作成 | 19 | 8.11 | YOLO による物体検出 | 65 |

1. はじめに

私の基礎演習では、様々な IT (情報技術) を知ることを目標にしています。コロナ感染症のために、我々は様々な面で行動の制約を受けました。それを補うために様々な IT が活用されることが期待されたのですが、日本はどうやら IT だめな国だったらしい、ということが明らかになりました。仕事がなくなり収入がなくなった方にお金を支給したいと思ったが、結局そういう方々を特定することができず、国民全員にお金を支給しました。その作業にもかなり時間がかかりました。他の国では申請してから一週間もかからずに支給された、と言う話もありました。コロナの感染拡大の状況も、保健所から Fax で送られてきた情報を手入力していると言う有様でした。その後オンラインで感染者情報を現場で入力することができるようになりましたが、入力すべき情報があまりにも多く、感染者の対応に忙しい現場ではとても入力する余裕がない、と言う状況も招きました。

IT を全て学ぶのはとても無理です。細かいところや難しいところは、理系の専門の方々にお任せするしかないでしょう。ただ「入力すべき情報が多すぎて使えないシステム」を作らせたのは理系の専門の方々ではないでしょう。現場を知らない、IT を知らない、そう言う人たちが大金を投じて役に立たないシステムを作らせる、と言うのは実は日本のあちこちで見られる風景です。

IT を利用して色々簡単にできる、ということを理解し、さらに自分で IT システムを作る人を育てたいと考えています。ささやかなシステムでも自分で作ることができれば、役に立つものか、使いやすいものかなどの検証ができます。そういうことができる人が増えれば、この国ももっとましになるのではないかと思います。基礎演習では IT で様々なことができることを学びつつ、「プログラミング基礎」などで IT システム作りの基礎を学んでもらいます。そして来年度の「展開演習」では実際に IT システムを作ります。案外簡単にできるものだなあ、と感じてもらえれば良いのですが。

技術の細かいところや難しいところは、理系の専門の方々にお任せするとして、代わりに磨いてほしいのは「利益」に対する感覚です。前世紀の末の頃の日本の大企業は世界的にも大企業でしたが、現在では見る影もありません。日本の大企業は儲けられなくて成長しない会社ばかりになってしまいました。利益は「売上」から「変動費」と「固定費」を引いたものです。まず利益にプラスとして関係する売上は、お客の数に比例すると言う考え方はだめです。つまりお客の数が 2 倍になれば、売上も 2 倍になるだろうというビジネスモデルでは足りません。Amazon や楽天などのオンラインモールでは、お客が増えると、売れるので出店も増えます。店が増えれば一人のお客が買うものも増えます。よってお客の数が 2 倍になれば、売上は 2 倍以上になります。この時注意しなければいけないのは、出店したお店は売上が 2 倍にしかならないところです。むしろ店が増えて同業者が増えると、売上の伸びが悪くなるでしょう。よって Amazon や楽天に出店して喜んではいけません。

売るのが情報だと変動費がほぼゼロになります。SNS の類はお客が情報を持ち込んでくれるので調達費もかかりません。一方パクられやすいのも情報です。音楽や動画配信ではコピー防止のために色々苦労しています。ただデジタル情報は劣化しませんが、新鮮さは次第に失われます。今後はコピー防止より、新しい情報の提供を重視する方が良いのではないかと思います。ただ調達費が十分用意できる規模がないと無理ですが。

売るのが物であれば、お客によるコピーの心配はなくなりますが、大量生産可能なものだと海外からコピー商品がどっとやってくるかもしれません。コピーされない高品質の実現がまず無理ならば、大量生産は諦めましょう。お客の個別の要求に合ったものを提供するという形態を実現するには、3D プリンターのような様々なものが作れる機械や、オンライン受注のシステムなどが必要になってきます。大量生産品に比べると変動費が高くなりますが、それ以上に販売価格を上げたり、直接お客に提供して中間マージンを削減できればなんとかなるでしょう。

固定費の削減は昔からの大きなテーマでした。IT による自動化による人件費の削減は、工場などでは大きな成果がありましたが、事務的な面ではまだまだです。せっかく入った会社からリストラされるのは誰でも嫌なことです。自動化がリストラにつながるとなると、当然協力したくないでしょう。仕事が楽になり、早く帰れ

るようになり、給料が上がるような自動化でなければ成功しないと思います。

IT系の固定費としてはWebサーバーやデータベースサーバーなどの各種サーバーの維持費があります。大きな仕事には大きなサーバーが必要になり、費用もかかります。最近ではAmazon、Googleやマイクロソフトのクラウドサービスを利用することにより、仕事量に合わせた規模のサーバーを使えるようになりました。クラウドサービスは様々な内容のものがあり、料金も仕事量にほぼ比例するため変動費として扱えます。またシステム管理の部分も任せられるのも大きな利点です。

ITを技術の面から捉えるだけでは、理系の学生に対して現代マネジメント学部の学生に勝ち目はありません。他の授業で習った経営・会計・経済・法律・政治などの知識もうまく組み合わせて行ける人を育てたいと思っています。

2. Linux (コンピュータ管理ソフト)

単一機能の電子機器に組み込まれる超小型のコンピュータを除いたほとんどのコンピュータで、OS (Operating System) と呼ばれるシステム管理ソフトが動いています。OS はコンピュータの持つ資源 (resource: 例えば CPU、メモリ、ファイル、入出力機器) を管理し、そのコンピュータで動くアプリケーションソフト (アプリ) の要求に従い、管理している資源を提供します。パソコンでは Windows が最もよく使われ、それ以外では Unix 系の OS がよく使われています。Unix 系の中では Linux が一番よく使われており、スマホの Android も Linux の一種です。

Windows については大学に入る前から使う機会があったと思いますが、Linux に関しては Android のスマホぐらいしか縁がないと思います。そして実際 Android のスマホをさわっても、iPhone とあまり変わらないなあ、という印象を受けるだけでしょう。見た目で言えば Linux は様々です。Windows、iOS (iPhone の OS)、Android などはバージョンが決まればほぼ同じ見た目や操作方法になりますが、Linux は組み合わせ方によって Windows や iOS とよく似た形にもできるし、全くグラフィカルな機能なしでキー入力だけでコマンドを入れて使うような形にもできます。そのような自由度があるため、スマホでも使われるし、Web サーバーのような、ひたすら送られてきた URL に対してデータを返すサービスをするコンピュータでも使われるのです。

2.1 ユーザー登録

複数の人が同時利用する OS では通常利用する人を登録するユーザー登録が必要です。そして利用者ごとに様々な設定が行われます。システムの管理者であれば、何でもできる権限が与えられ、一般の利用者はシステム関係の設定変更や他の利用者のファイルに触れることができないように設定されます。この演習では mars^{*1} という名前のサーバーをみんなで共有して利用します。よって演習を受講する学生を登録するための Web ページを作成しましたので、それを利用してユーザー登録を行います。ブラウザで「<https://mars.mgt.sugiyama-u.ac.jp/AC/>」へアクセスすると図 2.1 のようなページが表示されます。

利用者登録受付

これはmarsの利用者登録を受け付けるものです。

希望する登録名などを入力して、申請ボタンをクリックしてください。登録名は最大8文字で以下のような文字から構成されるものです。

- 数字、アルファベット、アンダーバー(_)が使えます
- 漢字や大文字は使えません
- 先頭に数字は使えません

なお、既に登録されたものと同じ名前は使用することはできません。

| | |
|-----------|---|
| 登録名 | 登録済みの名前: _apt, _rpc, aimi, akane, anz, aoka, avahi, awa, backup, bin, cat, chiaki, chimaaa, colord, cups-browsed, cups-pk-helper, daemon, dhcpcd, dnsmasq, fuku, fwupd-refresh, games, gdm, geoclue, gnats, hinata, hplip, irc, iuo, k, kernoops, kfcn, kokoro, kuno, list, lp, lxc-dnsmasq, maho, mail, maki, man, messagebus, michiko, miki, miku39, miku, miyu5, miyu, miyuki, moeka, news, nm-openvpn, nobody, pink, polkitd, postfix, proxy, pulse, riko, ririka, root, rtkit, rukira, saned, sara, sddm, shuri, smmsp, smmta, snap_daemon, snapd-range-524288-root, speech-dispatcher, sshd, sync, sys, syslog, systemd-network, systemd-resolve, systemd-timesync, syunharu, takunon, tcpdump, tknr, tss, usbmux, uucp, uuid, whoopsie, www-data, xrdp, yoshi, ysa, yuma |
| 学籍番号 | ex. A18EA999 |
| 氏名 (漢字) | ex. 楢山 花子 |
| 氏名 (ローマ字) | ex. Hanako SUGIYAMA |

申請

by K.Miki 2019/09/11

図 2.1 ユーザー登録の画面 (2022/10)

*1 正式な名称は mars.mgt.sugiyama-u.ac.jp です。なぜ mars かと言えば既に venus があったからです。

ここで自分の希望する登録名、自分の学籍番号と氏名を入力します。登録名の衝突はあまり考えられませんので、無理に長い複雑なものにする必要はありません。以下は昔の卒論生が使用した登録名です。

aki ako aoki asaka asako bluecat chizuko fujipon fusae happy hiromi ishii itomaki kaori kikumi kitty18 kozue kyoko lovin machi maki mame masami mayo miho mika misa miwako miyabi miyuki momoko naochan natu oida okada rena rie risa saori sasai sasara shiori taki tomo tomoko toshiko toya yuki yukko yuko yunke

2.2 リモートデスクトップによる接続

実は mars 本体には通常マウスやディスプレイが接続されていません。もし接続されていても、複数の学生が一つのディスプレイを覗き込んで作業をするのは無理でしょう。Unix 系の OS は基本的に複数利用者の同時利用が可能になっていて、通常利用者はネットワーク経由で利用します。

昔はネットワーク経由でサーバーに接続し、キーボードでコマンドを入力し、その結果をまた文字出力で得るという形 (CUI: Character User Interface) のみでした。実は現在でもこのような形で管理されているサーバーは少なくありません。一方パソコンなどでは、アイコンやメニューなどを利用した形 (GUI: Graphical User Interface) でコンピュータ利用が一般的で、同様な形でサーバーの操作ができれば便利です。その代わり CUI であればネットワークを経由して文字のやり取りで済みましたが、GUI では画面の状態なども常時送らないとマウスポインターも動きません。

Windows では、WindowsXP 以降*2に「リモートデスクトップ接続」というアプリが標準でインストールされています。これは本来は Windows によるサーバーへの接続のためのものですが、サーバーとのやり取りの部分を実似て、Linux であれば xrdp というアプリをインストールすれば、Windows のパソコンから Linux サーバーに接続して使用することができます。さらにスマホ用の「リモートデスクトップ」アプリもマイクロソフト社から無料で提供されています。よって使いやすいとは言えませんが、スマホから Linux や Windows サーバーを使用することも可能です。

Windows10 では、「スタート」→「Windows アクセサリ」→「リモートデスクトップ接続」で起動することができます。Windows11 では、「スタート」→「すべてのアプリ」→「Windows ツール」→「リモートデスクトップ接続」で起動することができます。起動すると図 2.2 のようなダイアログが表示されるので、コンピュータのところに「mars.mgt.sugiyama-u.ac.jp」を入力し[接続]ボタンをクリックします。

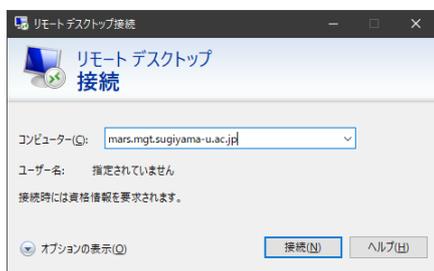


図 2.2 リモートデスクトップ接続の画面

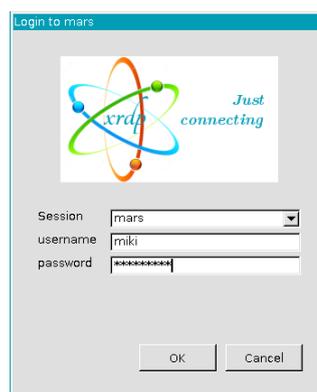


図 2.3 login の画面

無事に mars に接続できると、図 2.3 のような mars の認証のダイアログが表示されますので、mars での自分の登録名とパスワードを入力し[OK]ボタンをクリックします。

*2 WindowsXP、Windows vista、Windows7、Windows10、Windows11 の他、Windows NT 4.0 以降の Windows サーバー

すると図 2.4 のような Windows7 以前の Windows に似たデスクトップが表示されます。mars では Linux の中でも Ubuntu*³と呼ばれるディストリビューションの派生型の Lubuntu を使っています。Ubuntu は、GUI として GNOME*⁴を使用していますが、大勢で利用することから、より軽量で Windows に似ていると言われる LXQt*⁵を使用する Lubuntu にしています。

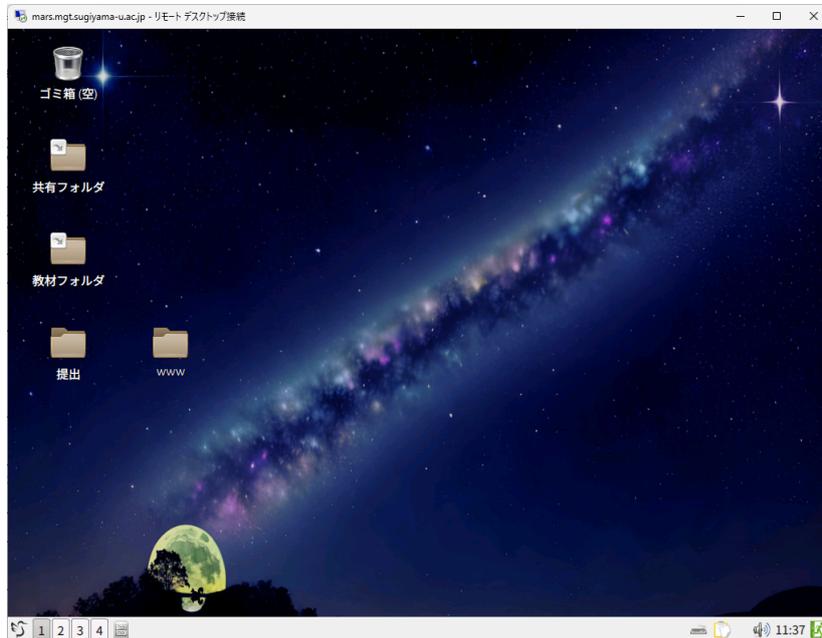


図 2.4 Lubuntu の画面

Lubuntu の基本的な操作は、

- 画面の左下角にあるスタートボタン? (🐧) をクリックするとメニューが出てくるので、そこで起動したいアプリを選択する。
- メニューの一番下の「ログアウト」を選択するか、画面右下角にある (🚪) をクリックすると終了することができる。ログアウトせずにリモートデスクトップ接続を終わると、次回接続した際にすぐに続きをすることができます。
- 漢字を入力する際には、キーボードの(半角/全角)キーを押すとかな漢字変換が起動される。再度同じキーを押せば元に戻る。

となっています。デスクトップにある「教材フォルダ」と「共有フォルダ」の内容は現マネ棟のパソコン演習室の同名のフォルダと同じ内容で、同じように使用することができます。

なお高解像度のノートパソコンなどでリモートデスクトップ接続を行うと、表示される文字が大変小さくなることがあります。このような場合は、まず図 2.2 の下部にある「オプション」、「画面」タブを順番にクリックし、「画面の設定」のスライダーを中間ぐらいにし、mars に接続した後にウィンドウの左上角のアイコン (🖥️) をクリックするとメニューが出るので、「拡大」で 150% などを選択します。

*³ <https://ja.wikipedia.org/wiki/Ubuntu> 「誰にでも使いやすい最新かつ安定した OS」を開発目標としている。最初から画像編集アプリやワープロソフト、ゲームなどが入っている。よってデスクトップ OS としてよく使われているが、サーバー用としても使われている。

*⁴ <https://ja.wikipedia.org/wiki/GNOME> Unix 系では X-Window という画面表示の基本システムがあり、その上に GNOME、KDE、LXQt などの様々なデスクトップ環境があります。

*⁵ <https://ja.wikipedia.org/wiki/LXQt>

2.3 パスワードの変更

システムが適当に設定したパスワードでは、入力しにくい、覚えにくいとか、パスワードを友人に知られてしまったような場合はパスワードを変更しましょう。ウインドウ左下角の「スタート」→「設定」→「パスワード変更」で図 2.5 のようウインドウが出ますので、現在のパスワード、新しいパスワードを 2 回入力して「パスワード変更」をクリックします。入力に間違いがなければ図 2.6 のようになりますので、「OK」をクリックします。



図 2.5 パスワード変更のウインドウ

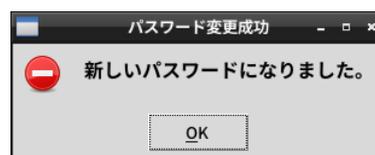


図 2.6 パスワード変更成功のウインドウ

2.4 インストール済みのアプリ

アプリのいくつかを簡単に紹介します。「アクセサリ：KCalc」は「アクセサリ」サブメニューにある「Calculator」という意味です。

- アクセサリ：FeatherPad 文書ファイル編集アプリ Windows の「メモ帳」を少し高級にしたもの。
- アクセサリ：KCalc 電卓アプリ
- インターネット：firefox ウェブ・ブラウザ Web ページ閲覧アプリ
- インターネット：Sylpheed メールを読み書きするアプリ
- オフィス：LibreOffice
 - Base データベースアプリ。Microsoft Access のようなもの。
 - Calc 表計算アプリ。Microsoft Excel のようなもの。
 - Draw 作図アプリ
 - Impress 発表用資料作成アプリ。Microsoft PowerPoint のようなもの。
 - Math 数式記述アプリ
 - Writer ワードプロアプリ。Microsoft Word のようなもの。
- オフィス：Okular PDF 表示アプリ
- グラフィックス：Drawing お絵かきアプリ Windows の「ペイント」のようなもの。
- グラフィックス：ScreenGrab 画面を画像ファイルに変換するアプリ
- システムツール：QTerminal Linux のコマンドを入力するためのアプリ

3. 文書清書システム

ここでは文書清書システムである \LaTeX (ラテフ) の使い方を説明します。 \LaTeX は同じく文書清書システムである \TeX (テフ) をより使いやすくしたのですが、ワープロソフトの生まれる前に誕生したもののなので、それでも大変使いにくいものです。イメージとしては、HTML のタグを文章に挿入して Web ページを作成するのに似ています*6。HTML のタグは何かして欲しいところに挿入します。 \LaTeX のコマンドも同様です。よって多少のコマンドは覚えなければなりません、レポートや論文ではそれほど多くのコマンドは必要ないので、あまり問題にならないでしょう。

同じコマンドは同じ結果をもたらすために、文書の見た目を揃えることができます。Word でその日の気分で見出しのフォントの大きさを変えたりしていると、簡単に統一の取れていない文書ができてしまいますが、 \LaTeX ではそのような問題を避けることができます。また HTML のタグと違い、 \LaTeX のコマンドはカスタマイズが可能です。標準の大きさや形が気に入らない場合は変更することができます。

システムの処理結果を示す場合、ブラウザに表示するのであれば HTML を使用し、体裁を整えるならさらに Style Sheet を使用するのが普通でしょう。紙に出す場合、プリンターごとに指定した文字を指定した場所に出す方法が異なるので、様々なプリンターに対応するのは困難です。そこでよく使われているのが処理結果を PDF の形にして出すと言う方法です。どのようなソフトで作成した文書も、PDF であればパソコンのプリンターで印刷できます。あとは処理結果をきれいな PDF の形にするところですが、処理結果に \LaTeX のコマンドを追加して清書し、その結果を PDF にすることができます。これならばブラウザに表示するのと余り変わらない手間、紙に印刷ができます。

3.1 Texmaker の起動

\LaTeX の原稿となるファイルは、Windows 付属の「メモ帖」ソフトでも入力が可能です。HTML のタグと同様に \LaTeX のコマンドも普通のキーボードで入力できる文字で構成されているからです。しかし、入力が終わればコンパイルという処理やその結果の表示などの操作が必要なので、これらを簡単に実行できるソフトが色々作られています。mars では Texmaker というソフトをインストールしていますのでこれを利用します。ウインドウの左下角の「スタート」→「オフィス」→「Texmaker」で起動できます。また拡張子が tex のファイルをダブルクリックしても、Texmaker の起動・読み込みができます。

\LaTeX はコンパイル処理の際に複数のファイルを生成します。文書を変換したファイルの他、目次や参照などのデータが別のファイルになります。ファイルがばらばらにならないように、文書ごとにディレクトリ(フォルダー)を作成することをお勧めします。

3.2 簡単な例

Texmaker を起動しただけでは文章の入力できません。「ファイル」→「新規作成」をすると入力できるようになります。簡単な例として以下の文章を入力してください。入力する際に \backslash は $\boxed{\text{\LaTeX}}$ (半角) のキーを押します。また \backslash で始まる横文字の部分(\LaTeX のコマンド)は「LaTeX」メニューに入れることができます。また \LaTeX のコマンドの最初の数文字を入れると、コマンドの候補が出てきます。複数の候補が出てきた場合は更に入力するか、矢印キーで選択してから $\boxed{\text{\LaTeX}}$ を押すと確定されます。

*6 HTML が Tex の真似をしたという方が正しいが。

```
\documentclass[10pt]{jsarticle}
\title{簡単な例}
\author{A24EA999 梶山花子}

\begin{document}
\maketitle

\section{はじめに}

これは大変簡単な例です。文章中の改行は無視されるので、入力の際には
適当に改行してください。1行空けると段落の終わりともみなされます。

これは2つ目の段落です。先頭は勝手に字下げされるので空ける必要はありません。
\end{document}
```

日本語の部分はこの通り入れなくても大丈夫です。無事入力ができたら「ファイル」メニューの「名前をつけて保存」で保存します*7。そして[F1]のキーを押すとコンパイルの処理が行われて、問題がなければPDF形式の結果ファイルが生成されて、表示されます。

誤りがあった場合には、画面の下に残念ながら英語でメッセージが表示されます。誤りのあった行番号が示されるのでそれを参考に修正をしてください。なお、修正のあとは上書き保存をしなくても、[F1]のキーを押すとコンパイル処理の前に上書き保存をしてくれます。

3.3 印刷する方法

通常インターネットのどこからでも mars にアクセスすることができます。つまりどこからでも文書作成が可能です。ところが作成した文書を印刷するとすると、mars の側にあるプリンターから出てくるのでは大抵の場合困ります。現在自分のいる場所に近いプリンターから出てくるのが望ましいのですが、それがどれなのかは mars には分かりません。コンパイル処理された文書はPDF形式のファイルになっているので、これを自分の使っているパソコンにコピーして、パソコンから印刷することによって、身近なプリンターから出すことができます。

1. PDF形式の文書ファイル(～.pdf)のアイコンを右クリックする。
2. 「コピー」を選択する。
3. パソコンの適当な場所(例えばデスクトップの空いたところ)で右クリックする。
4. 「貼り付け」を選択すると、文書ファイルがパソコンに送られる。卒論のような大きなファイルや画像を多数含むファイルは多少時間がかかるかもしれません。
5. パソコン上の文書ファイルをダブルクリックして開き、そこで印刷の指示をする。

3.4 LaTeX のコマンド

基本的なコマンドの一部を紹介します。なおこの部分は鈴木裕信が書いた「ひろのぶの LaTeX 入門 Version 0.23」を元はかなり省略したものになっています。

LaTeX には局所的なコマンド(command)と広域的な環境(environment)があります。コマンドは\で始まります。そして{}や[]で囲まれた引数を取ることができます。たとえば、節の見出しのコマンドは\sectionであり、この章の原稿には、\section{文書清書システム}と書いてあります。

一方環境は\begin{name}と\end{name}で囲まれた範囲が、nameと言う規則にすべて従うという感じになります。

*7 拡張子として.texが必要ですが自動的に補われます。

3.5 使用できる文字

文中で使用できる文字は全角文字、半角の英数字や記号などです。ただし、次の 10 個の特殊文字は、 \LaTeX の命令に使用されます。ですからそのまま使用することはできません。

\$ % & _ { } ~ ^ \

\ 以外は \#, \\$ とすることによって文字として表現できます。 \ は次のような方法で表現します。

\backslash ⇒ \

\LaTeX はテキスト中に空白があっても自動的につめてしまいます。意図的に空白を挿入する時は、 \ する必要があります。改行の場合は \\ を使います。マイナス記号 (-) は特別な働きをします。連続して書くことによってハイフンなどの線 (- - -) を引くことができます。

3.6 文章のスタイル

一番最初は必ず \documentclass[option]{style} で始めなければなりません。そして文章の本体は \begin{document} と、 \end{document} で囲まれた範囲内に書きます。

文書のスタイル (style) の標準的なものには次のようなものがあります。

| | 欧文 | 和文 (旧) | 和文 | 和文 (縦書き) |
|-----|---------|----------|-----------|----------|
| 論文 | article | jarticle | jsarticle | tarticle |
| 本 | book | jbook | jsbook | tbook |
| 報告書 | report | jreport | - | treport |

「論文」スタイルは最もよく使われます。標準では用紙の下部の中央にページ番号が付きます。このテキストもこのスタイルに手を入れて使っています。 \section のレベルから書くことができます。

「本」のスタイルでは、奇数ページの上部 (ヘッダ) では節 (セクション) の番号とセクション名が左側、ページ番号が右側に出ます。偶数ページヘッダでは右側に章 (チャプタ) 名、ページ番号が左側に出ます。章の最初のページが奇数になるようにページが送られます。章立ては必ず \chapter から始めなければなりません。

「報告書」はレポートの形式です。ページ番号は用紙の下部の中心にくるような形になります。章立ては必ず \chapter から始めなければなりません。もし、 \section レベルから始めた場合、エラーとなります。

オプション (option) はスタイルの補助的な役割をします。基本となるフォントサイズの変更や見開きに変更する事ができます。

| | 指定なしの場合 | 指定できるもの |
|------------|--------------------------------------|---|
| 文字の大きさ | 10pt | 9pt、10pt、11pt、12pt、14pt、17pt、21pt、25pt、30pt、36pt、43pt |
| 用紙サイズ | A4 サイズ | a4paper、a5paper、b4paper、b5paper |
| 本文の段組 | onecolumn (一段組) | onecolumn、twocolumn |
| 見開きの設定 | 本のスタイルでは twoside、 それ以外では oneseide | oneside、twoside |
| 表題を別ページにする | 論文スタイルでは本文と同じページに | titlepage |

3.7 タイトルページ

タイトルページは \maketitle があらわれた時点で作成されます。よってその前に \title や \author を使って表題や著者を指定する必要があります。 \date は指定しなければ、コンパイルを行った日時が使われ

ます。

```
\title{HOW TO USE \LaTeX}
\author{Hironobu Suzuki}
\date{4 Oct 1988}
.....
\maketitle
```

⇒

HOW TO USE L^AT_EX
Hironobu Suzuki
4 Oct 1988

タイトルページに要約に付ける時は次のようにします。もちろん `\maketitle` コマンドより前にこれを使う必要があります。

```
\begin{abstract}
This article is for the beginner.
This abstract is printed onto top page.
\end{abstract}
```

⇒

Abstract
This article is for the beginner. This
abstract is printed onto top page.

3.8 文章構成

文章の構成で、章や節を表すものとして、次のようなものが用意されています。

| | | |
|-----------------------|-----------------------------|----------------------------|
| <code>\part</code> | <code>\section</code> | <code>\paragraph</code> |
| <code>\chapter</code> | <code>\subsection</code> | <code>\subparagraph</code> |
| | <code>\subsubsection</code> | |

各々 `\subsection{文章構成}` のように見出しの文字を指定します。通常の `jsarticle` スタイルでは、節 (section) の中が項 (subsection) に分かれ、さらにその中が部分項 (subsubsection) に分かれます。 `jbook` や `jreport` は章 (chapter) の中が節に分かれ、以下 `jarticle` と同様に細かく分かれて行きます。

見出しの番号は `LaTeX` が自動的に付けてくれます。見出しの文字の大きさなども自動的に設定されます。 `\section` を飛ばして `\subsection` を使用すると番号が変になるのでご注意ください。

3.9 空白

半角の空白は複数あっても、1つの空白と見なされます。逆に全角の空白は目に見えませんが、行の前後に残っていると歯抜けの段落ができたりするので注意が必要です。

意図的に空白を挿入したい場合は `\□` とします。これは `LaTeX on UNIX` のような書き方をしたとき `LATEX on UNIX` というような結果になってしまいます。このような結果を避けたい時は `LaTeX\□on□UNIX` とすれば `LATEX on UNIX` となります。

`\hspace` を使用すると、きっちり指定した幅だけ空けることができます。ただしその空白がちょうど行の先頭や末尾に来たときは無視されますのでご注意ください。

```
1cm \hspace{1cm} space.
1cm \hspace{1cm} space.
1cm \hspace{1cm} space.

This command can accept minus
paramater like this \hspace{-1cm}////
```

⇒

```
1cm      space.  1cm      space.  1cm
space.
This command can accept minus paramater
like\this
```

3.10 文字のスタイル

文字の大きさは直接ポイント数を与えて変えるような野蛮なことはできません。 `large` や `small` などのコマンドによって文字のポイント数が変わりますが、原稿の基準フォントのポイントサイズに合わせて、相対的に

large であり、small であります。このように相対的に指定する事により、後になって基本となるサイズを変更した場合にも、本文の指定を変更する必要がなくなります。

文字の大きさの有効範囲は同一ブロック内です。ブロックとは、環境の中や { と } で囲んだ範囲の事です。例えば、{\Large abc}d と行なうと abc が大きくなり d は元のサイズのままです。実際に行なってみると abcd となります。

通常の文書で頻繁に文字の大きさを変えると見やすさよりも見にくさを助長するだけなので、次の指定はあまり使う機会はないでしょう。

| | | |
|-----------|------------------|----------------|
| 文字 \Huge | 文字 \huge | 文字 \LARGE |
| 文字 \Large | 文字 \large | 文字 \normalsize |
| 文字 \small | 文字 \footnotesize | 文字 \scriptsize |
| 文字 \tiny | | |

英数字の文字のスタイルは色々ありますが、全角に関しては太字と下線のみが有効です。影響を与える範囲などは文字の大きさと同じです。

```
\textbf{Bold type style. 全角は太字で  
す。}  
\textsf{Sans serif type style. 全角は  
同じです。}  
\textsl{Slanted type style. 全角は同じ  
です。}  
\texttt{Typewriter type style. 全角は  
同じです。}  
\textsc{Small caps type style. 全角は  
同じです。}  
\emph{A emphasized line. 全角は太字で  
す。}  
\underline{underlined text} \underline{  
下線付き文字です。}
```

⇒

```
Bold type style. 全角は太字です。  
Sans serif type style. 全角は同じです。  
Slanted type style. 全角は同じです。  
Typewriter type style. 全角は同じです。  
SMALL CAPS TYPE STYLE. 全角は同じです。  
A emphasized line. 全角は太字です。  
underlined text 下線付き文字です。
```

下線はコマンドなので指定の仕方が異なりますのでご注意ください。

3.11 行、段落

原稿の文字列は自動的に詰められ改行されたりします。空行は段落の切れ目を意味しますが、複数の空行は 1 行分の空行と解釈されます。段落の先頭は 1 字下げがなされるので、原稿では字下げは不要です。

文字列や単語の途中で改行が入ってしまった場合は \mbox を使います。例えば、*Segmentation fault* の *Segmentation* と *fault* を離したくない場合は \mbox{*Segmentation fault*} とします。

左につめて書きます。ただし、前の行は空白行です。

```
連続した行です。  
連続した行です。  
連続した行です。  
連続した行です。  
連続した行です。
```

⇒

左につめて書きます。ただし、前の行は空白行です。

```
連続した行です。連続した行です。連続した  
行です。連続した行です。連続した行です。
```

3.12 行の空き方

LaTeXでは段落、表や画像などがくっつくことがよくあります。これらの間を空けたいときは、`\smallskip`、`\medskip`、`\bigskip`を使用します。実際の間隔はページのレイアウトに左右されますので、同じコマンドでもいつも同じ間隔が空くとは限りません。

行間や段落間の間隔を調整する直接指定する方法として`\vspace`などのコマンドがありますが、既に書式が決まっているものに無理矢理合わせるような場合以外は使わない方が無難ですので例は省略します。

前の行は空白行です。

```
\smallskip
前の行は空白行です。
```

```
\medskip
前の行は空白行です。
```

```
\bigskip
前の行は空白行です。
```

⇒

前の行は空白行です。

前の行は空白行です。

前の行は空白行です。

前の行は空白行です。

3.13 中揃え・右揃え

通常のワープロソフトを使用する際でも、中揃えや右揃えは空白を使って位置を調節するのではなく、そういう指示で行うのが常識です。LaTeXでは次のようにして中揃えを実現します。

```
\begin{center}
ここにあるものは、中央に寄せられます。
```

中央

```
\end{center}
```

⇒

ここにあるものは、中央に寄せられます。

中央

一方 `flushright` を使用すると右揃えになります。

```
\begin{flushright}
ここにあるものは、右に寄せられます。
```

右寄り

```
\end{flushright}
```

⇒

ここにあるものは、右に寄せられます。

右寄り

3.14 箇条書

箇条書の書式として LaTeX では3種類の方法があります。先頭に●をつけるもの、先頭に番号を振るものと、先頭にラベルを指定できるものです。それぞれ、`itemize`、`enumerate`、`description`です。

```
\begin{itemize}
\item 各々の行はあたまたにポチを付けられてな
らんでいるでしょ。
\item
書き出しは次の行でも大丈夫です。
\end{itemize}
```

```
\begin{enumerate}
\item この行の頭には数字がついているはず
です。
\item ほらね、2行目でしょ。
\end{enumerate}
```

```
\begin{description}
\item [ラベル] この行の頭にはラベルが
ついているはずですよ。
\item [Label] ラベルは bold 体になってい
ます。
\end{description}
```

⇒

- 各々の行はあたまたにポチを付けられてな
らんでいるでしょ。
- 書き出しは次の行でも大丈夫です。

1. この行の頭には数字がついているはず
です。
2. ほらね、2行目でしょ。

ラベル この行の頭にはラベルがついている
はずですよ。

Label ラベルは bold 体になっています。

3.15 そのままの形

`verbatim` はテキストを入力したそのままの形で表すための環境です。例えば、`%\}` はそのまま打っても $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ では正しく表示してくれません。また HTML のタグには必須の「`< >`」もそのままでは「`j i`」のように化けてしまいます。プログラムのリストをそのまま引用するような場合、この環境を使用します。

```
\begin{verbatim}
次のキャラクターは通常、特殊文字として扱わ
れるので表示できない。
バックスラッシュを使うか verbatim を使うか
どちらかである。
\end{verbatim}
```

⇒

次のキャラクターは通常、特殊文字として扱わ
れるので表示できない。
バックスラッシュを使うか `verbatim` を使うか
どちらかである。

行中にこれらの特殊文字を組み込みたいときは、`\verb` を使用します。`\verb+<??###$$>+` とすると、`<??###$$>*`⁸と出力されます。

3.16 表

$\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ では、表のようなものを作るのに、`tabbing`、`tabular`、`array` という環境が利用できます。それぞれ想定された用途があり、特徴がありますがここでは `tabular` のみ説明をします。HTML の表も様々なタグが多数必要で面倒でしたが $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ はそれ以上に面倒です。ただ `Texmaker` の「作成支援」メニューの中に「表の速成」というものがあります。とりあえずこれで表を作成し、後で微調整すると言うやり方も可能です。

`tabular` の引数の中での命令の意味は、`l` は左詰め、`r` は右詰め、`c` は中央に各項目をそろえます。`p{wd}` は列の幅を `wd` に指定 (たとえば `p{1cm}`) することにより、固定することができます。

| をつけると垂直方向の線が書けます。`\`の後の `\hline` は水平方向の線を引きます。`\cline{i-j}` は `i` 列から `j` 列までの水平方向の線を引きます。

⁸`\verb` の次の記号 (本文の例では `+`) がそのままの形で出力される部分の始まりと終わりを示します。`+` 以外の記号も使用可能です。

```

\begin{tabular}{|l|l|c|r|}
\hline
sra & sra-b1-net & 12098 packets & \\
\cline{2-3}
& sra-2-net & 290873 packets & \\
\hline
ntt & sun-loop-back & 2839 packets & \\
\cline{1-1} \cline{3-3}
etl & & 287 packets & \\
\hline
\end{tabular}

```

⇒

| | | |
|-----|---------------|----------------|
| sra | sra-b1-net | 12098 packets |
| | sra-2-net | 290873 packets |
| ntt | sun-loop-back | 2839 packets |
| etl | | 287 packets |

一つ一つの項目欄は`\multicolumn{n}{pos}{item}`によって調整することができます。n は使用する列数、pos は項目の水平方向の位置、item は項目の文字列です。引数である pos は現在すでに設定されているものを無効とし新たに設定します。この時有効となる引数 pos は `l r c | p` からなるものでなくてはなりません。

```

\begin{tabular}{|l|l|l|r|}
\hline \hline
{\em team } & & & \\
\multicolumn{2}{c|}{\em driver} & & \\
\hline
マクラーレン & セナ & プロスト & \\
ロータス & ピケ & 中島 & \\
\hline \hline
\end{tabular}

```

⇒

| <i>team</i> | <i>driver</i> | |
|-------------|---------------|------|
| マクラーレン | セナ | プロスト |
| ロータス | ピケ | 中島 |

3.17 脚注の入れ方

ページの下の部分に脚注を入れることができます。本文の補足的な情報をここに入れます。例えば URL^{*9}などは参考文献として挙げて良いですが、それほど重要でもなければ脚注に入れた方が本文が読みやすくなります。脚注の参照番号が入るところに `\footnote{...}` を入れます。「...」の部分に脚注の文章が入ります。

3.18 数式

数学モード (math mode) は数式を表現するために使用するモードです。TeX は元もと数学の本を美しく作るために生まれたために数式の表現は大変得意です。しかし、とりあえずは x^2 、 $\sqrt{2}$ と $\frac{2}{3}$ ぐらいが書ければ十分でしょう。

文中とは独立して数式を示したいときには、`\begin{math} ... \end{math}` を使います。文中に数式を入れる場合には `$...$` を使います。例えば、`$ \int \sin(2x+3)\cos(x)^2 dx $` と記述すると、 $\int \sin(2x+3)\cos(x)^2 dx$ のように出力されます。

数式中での英字は xy というように斜体になります。一方関数名などは例えば、`log` は通常の字体でなければなりません。そこで `\log` を使用します。例えば、`$\log x^y = y \log x$` とすれば $\log x^y = y \log x$ となります。

| | | | |
|-------------|------------------------|---------------|--------------------------|
| \pm | <code>\pm</code> | \times | <code>\times</code> |
| A_b^c | <code>A_{b}^{c}</code> | $\frac{x}{y}$ | <code>\frac{x}{y}</code> |
| \sqrt{A} | <code>\sqrt{A}</code> | $\sqrt[b]{A}$ | <code>\sqrt[b]{A}</code> |
| $gcd(x,y)$ | <code>\gcd(x,y)</code> | $\log(x)$ | <code>\log(x)</code> |
| $\max(x,y)$ | <code>\max(x,y)</code> | $\min(x,y)$ | <code>\min(x,y)</code> |
| A^b | <code>A^{b}</code> | A_b | <code>A_{b}</code> |

*9 Uniform Resource Locator: 例えば https://ja.wikipedia.org/wiki/Uniform_Resource_Locator

3.19 画像の入れ方

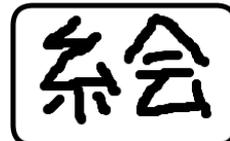
まずいちばん最初の`\documentclass`と`\begin{document}`の間に次のパッケージの設定を入れます。パッケージは LaTeX の機能を拡張するためのもので、これ以外にも様々なもの^{*10}があります。

```
\usepackage[dvipdfmx]{graphicx,color}
\usepackage{bmpsizer}
```

そして画像を入れたいところに次のようなコマンドを入れます。

```
\includegraphics[width=3cm]{sample.png}
```

⇒



この例の場合、「sample.png」という画像ファイルの内容が、「width=3cm」によって幅 3cm に整形されて取り込まれます。また「scale=0.5」のように相対的な大きさ (倍率) で指定する事もできます。例えば画面から同じように取った複数の画像は、同じ倍率で指定した方が良いでしょう。画像ファイルの形式は、JPEG と PNG のものが可能です。また PDF 形式の文書もこれで取り込むことができます。

画像ファイルは様々な方法で作成することが可能ですが、最終的に原稿のファイルがあるところへ持ってくる必要があります。パソコンまで持ってこれれば「印刷する方法」で PDF ファイルをパソコンに送った方法の逆をすれば、mars に画像ファイルをコピーすることができます。また mars にも画像を編集するソフト「Drawing」^{*11}があるので、Windows の「ペイント」のようにお絵かきも可能です。

スマホで撮影した画像を mars に持ってくる場合は、メールに添付して mars へ送る方法があります。またパソコンの画面を画像として取り込む事も可能です。Windows の場合(Print Screen)キーを押すと全画面が取り込まれますので、mars の「Drawing」を起動して貼り付けるか、Windows の「ペイント」を起動して貼り付けます。アクティブなウィンドウのみを取り込む場合は(Alt)を押しながら(Print Screen)キーを押します。mars の中のものを画像にする場合はこれらのキーを利用しても良いですが、「ScreenGrab」^{*12}と言うソフトが入っていますので、こちらを利用すると直接画像ファイルになります。

3.20 図表の参照

文書の中に図や表が一つしかない場合はどのような書き方をしても間違いがありませんが、複数の場合はどの図表を指しているのかわかりにくくなります。また図や表のような比較的大きなものの場所を厳密に指定してしまうと、図表がページの残りに入り切らない時に文中に大きな空気ができてしまうこともあります。このような問題を生じないように図や表は次のようにしましょう。

図の場合

画像の場合は次のようにします。

^{*10} 例えば <https://www.biwako.shiga-u.ac.jp/sensei/kumazawa/texindex.html> には 300 種類以上のパッケージが紹介されています。

^{*11} 画面の左下の「スタート」→「グラフィックス」→「Drawing」と選択して起動できる。

^{*12} 画面の左下の「スタート」→「グラフィックス」→「ScreenGrab」と選択して起動できる。

```

\begin{figure}[htbp]
\centering
\includegraphics[width=3cm]{sample.png}
\caption{図の例}
\label{zu-rei}
\end{figure}

```



図 3.1 図の例

最初の行の「htbp」は図をどこに置くかの指定です。「原稿の位置のとおりに表示 (here)」、「ページの上の方に表示 (top)」、「ページの下の方に表示 (bottom)」、「別のページに表示 (page)」の順番に検討するという意味です。`\centering` は図の位置を中央揃えにする指示です。`\caption` で図の見出しを指定し、`\label` でこの図を参照する際の名前を指定します。文中でこの図について触れる箇所に「`\ref{zu-rei}`」を入れると、「図 3.1」のように表示されます。また「`\pageref{zu-rei}` ページ」を入れると「17 ページ」と言うように図のあるページも表示できます。

図を同じ場所に複数並べる際には `minipage` 環境を使用する必要があり、結構複雑になります。

表の場合

表の場合も `tabular` 環境を `table` 環境の中に入れます。`figure` の部分が `table` になるだけで、後は図の場合と変わりません。

```

\begin{table}[htbp]
\centering
\caption{表の例}
\label{hyou-rei}
\begin{tabular}{|l|l|l|}
\hline
aaa & bbb & ccc \\
\hline
xxx & yyy & zzz \\
\hline
\end{tabular}
\end{table}

```



表 1 表の例

| | | |
|-----|-----|-----|
| aaa | bbb | ccc |
| xxx | yyy | zzz |

表 1 のように `caption` コマンドを先に使用すると、表の上に見出しを付けることができます。図と表の番号はそれぞれ独立してカウントされます。

3.21 参考文献リスト

文書内で、他の本や Web ページなどに掲載された事実や意見などを引用した場合、どこに掲載されていたものか示さないと問題になることがあります。また自分のオリジナルでない意見などは、誰によるものかを明らかにした方が信じてもらえます。

簡易的に出典を示すには既に出てきた脚注を使用しても良いでしょう。もう少しちゃんとしたやり方としては、次のように `thebibliography` 環境の中で `bibitem` コマンドで文献を示し、`cite` コマンドでそれを参照します。

参考文献 `\cite{latex}` を参照のこと。

```
\begin{thebibliography}{99}
  \bibitem{latex} Leslie Lamport 著、Edgar Cooke・倉沢良一監訳、
  「文書処理システム  $\LaTeX$ 」 p.191-192、株式会社アスキー、1990 年
\end{thebibliography}
```

`thebibliography` 環境で「99」を指定しているのは、参考文献の数が 2 桁という意味です。大抵の場合 99 で十分でしょう。`bibitem` コマンドで書く内容は、図書の場合、著者名、図書名、出版社、出版年が最低限必要です。これによって次のように示されます。ただし `cite` コマンドで表示される番号は 2 回コンパイルしないと出てこない所以需要が必要です。参考文献 [1] を参照のこと。

近年は多くの有益な情報がインターネットの Web ページから得られます。これらを参考文献として挙げる場合、長く記号の多い URL の扱いに困ります。そのような場合は、ファイルの最初の所に「`\usepackage[dvipdfmx]{hyperref}`」を追加し、文中で URL を書く際に「`\url{https://...}`」のようになると文字化けなく出るようになります。さらに PDF では、URL の部分がリンクになります。

参考文献

- [1] Leslie Lamport 著、Edgar Cooke・倉沢良一監訳、「文書処理システム \LaTeX 」 p.191-192、株式会社アスキー、1990 年

4. 三次元モデルの作成

ここでは、三次元モデル作成ソフトである Blender^{*13} ver.4.02 の使い方^{*14}を説明します。これで作成した三次元モデルは、3D プリンターで実物を作成したり、AR(拡張現実)を利用して現実世界へ投影することができます。三次元モデルを作成するためのソフトは様々なものがあります。Blender は Windows だけでなく Linux などでも使用可能な高機能なフリーソフトです。

Blender は高機能のため様々なことができますが、逆に機能が豊富すぎて困る場合も少なくありません。例えばメニューも多数あり、かつ多段階になっているのでなかなか望みの機能にたどり着けません。また二次元のお絵描きと比べると三次元モデルの作成はかなり複雑と言う点があります。お絵描きソフトで絵を描く場合、二次元のディスプレイの上に見えたものがそのままお絵描きの結果となります。三次元の場合、二次元のディスプレイに見えるものはある方向から見た場合の姿ですので、別の方から見ると全然予定と異なる形になっていることも少なくありません。また色についてもお絵描きでは、色を選択して塗りつぶすという感じですが、三次元では材質(マテリアル)の色(ディフューズ)を設定し、さらに鏡面反射(スペキュラー)する色、反射する割合なども設定します。光線の位置や強さも別に設定することにより、他の物体の映り込みなどは自動的に計算されます。逆にお絵描きでは映り込みなどは自分で描き込まなければいけません。

さらに三次元モデルでは動かす事も可能になっています。アニメの登場人物を三次元モデルで作成する際には、モデルの内部に骨や関節を設定し、体の各部分をシーンに合わせて動かせるようにします。お絵描きでアニメを作成する場合は、少しずつ動いたシーンを描くこととなりますが、三次元の場合は一つのモデルの変形で対応できます。さらに MikuMikuDance^{*15}のような動画作成ソフトでは、最初と最後の姿勢を指定すると、その間の動きを自動的に補間してくれますが、二次元の絵で同様のものはありません。

4.1 Blender の起動方法

パソコンで Blender を使用する場合は、そのパソコンに Blender を自分でインストールする必要があります。mars には既にインストールしてありますので、すぐに使うことができます。ただし複数の人が同時に使

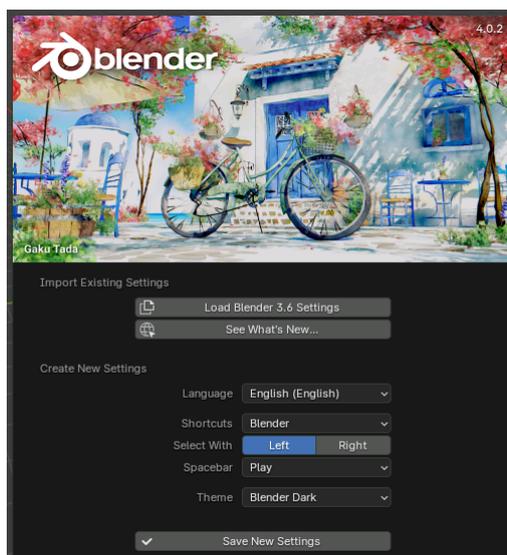


図 4.1 Blender の初期設定画面

^{*13} <https://www.blender.org/>

^{*14} 他のソフトでもよくあることですが、バージョンが異なると操作法が変わってしまうので、検索して使い方を探す場合はどのバージョンの説明なのかに注意しましょう。Blender は ver.2.80 からかなり変わりました。

^{*15} <https://sites.google.com/view/vpvp/>

うと処理が追いつかなくなって遅くなる可能性があります。mars ではウィンドウの左下角の「スタート」→「グラフィックス」→「Blender」で起動できます。起動するとウィンドウの中央に図 4.1 のようなパネルが表示されます。blender を初めて起動した際にはここで、「Language」が「English (English)」になっているので、「Japanese (日本語)」に変更します。そして「新しい設定を保存」をクリックします。次回からはパネルの内容が少し変わりますが、パネルの外をクリックしてすぐこのパネルを消しても大丈夫です。

パネルを消すと図 4.2 のような Blender の画面が出てきますが、かなり複雑です。中央のオブジェクトの表示画面 (3D ビューウィンドウ) の周りを様々なメニューやボタンが取り囲んでいます。

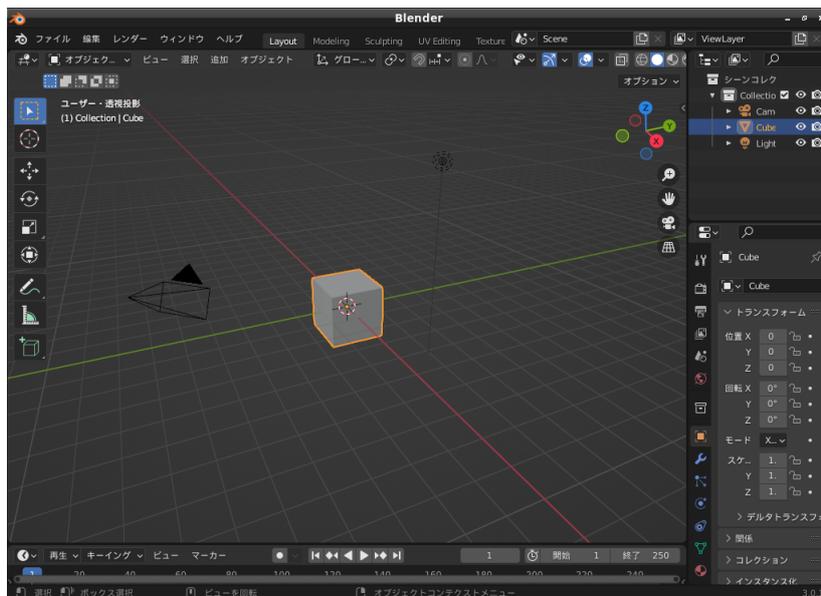


図 4.2 Blender の初期画面

Blender で作成する三次元モデルを構成するものをオブジェクトと呼びます。各オブジェクトは平面で構成されています。曲面ではありません。球のような丸いものは、実用上問題のないレベルまで細かく分割した平面にします。

Blender では様々な操作をメニューで選択して行うことができますが、キー入力でもかなりの操作を指示することができます。メニューで一つ選択するよりもその方が早いのでキー入力で行う方を勧めますが、マウスポインターが適切な場所がないとうまく行きませんのでご注意ください。また操作を間違えたらすぐ **(Ctrl)+Z** で戻しましょう。

Linux 版の Blender ではかな漢字変換が使用できないという問題点が以前から残っています。スタートメニューの「アクセサリ」の中の「FeatherPad」を起動して、そこへ入力したものをコピーで移すという形で対処します。

4.2 シーンに対する操作

全体をシーンと呼びます。シーンには、複数のオブジェクトだけでなく、照明やカメラが含まれます。マテリアルにいくら色を設定しても、発光する設定がなく、照明がないと真っ黒です。同じ色でも照明の当て方によって明るさが変わりますし、影も生じます。よって Blender で綺麗なオブジェクトの画像を作成したい場合は、照明をどのように当てるかは大きな問題となります。一方オブジェクトを取り出して 3D プリンターで造形する場合は、Blender の外で様々な光に当たる訳ですから、照明の設定は不要です。カメラの位置や向きなどの設定もオブジェクトの画像を作成したい場合の際は重要ですが、オブジェクトを外へ取り出す場合は関係ありません。

4.2.1 寸法の単位の設定

オブジェクトを 3D プリンターで実物にする場合や AR(拡張現実) で設定した大きさで表示したい場合、まず単位を設定する必要があります。右側のパネルのところで図 4.3 のように設定をします。まず①の部分が違うものになっていたらクリックして「プロパティ」を選択します。次に②をクリックし、③の「単位系」が「メートル法」になっているのを確認します。そしてその下の「単位の倍率」を「0.001」に変更してから④の「長さ」を「ミリメートル」にします。大きめのオブジェクトを作成するならば「単位の倍率」を「0.01」に変更してから④は「Centimeters」(cm) でも良いでしょう。

4.2.2 シーンを見る向きの変更

マウスポインターを中央のオブジェクトの表示の部分へ持って行って、テンキーの数字を押すと表 2 のようにオブジェクトの見える向き(視点)が変わります。キーボードのアルファベットの上に横一列に並んでいる数字のキーではだめなのでご注意ください。テンキーを押す際に(Ctrl)キーを押しながらだと回転などの向きが逆になります。(Shift)キーを押しながらだと選択対象基準で回転などをできるようになります。視点を色々変更していると、時々オブジェクトが何も見えなくなります。そういう場合はテンキーの「.」を押すと見えるようになります。

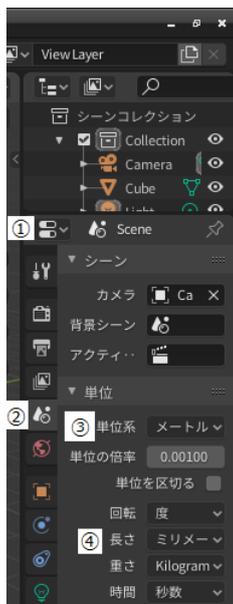


図 4.3 寸法の単位の設定

表 2 テンキーによる視点の変更

| キー | 視点 |
|----|-----------------------|
| 2 | 下が見える方向に 15° 回転 |
| 8 | 上が見える方向に 15° 回転 |
| 4 | 左が見える方向に 15° 回転 |
| 6 | 右が見える方向に 15° 回転 |
| 9 | 180° 回転 |
| 1 | 正面からにする |
| 3 | 右正面からにする |
| 7 | 真上からにする |
| 0 | カメラからの視点にする ⇔ 元にもどす |
| 5 | 投影方法の切り替え：透視投影 ⇔ 平行投影 |
| . | オブジェクトの全てが見えるようにする |
| / | 選択しているオブジェクトが見えるようにする |

マウスのホイールで視点からの距離を変えることができます。近くにすればオブジェクトが大きく見え、遠くにすれば小さく見えます。また(Shift)を押し、マウスのホイールを押しながらマウスを動かすと視点を上下左右に動かすことができます。

4.3 オブジェクトに対する操作

クリックしてオブジェクトを選択した上で、(Tab)を押すと「オブジェクトモード」と「編集モード」の切り替えができます。「オブジェクトモード」では、次に述べるようなオブジェクトの位置などオブジェクト全体の操作ができます。またオブジェクトの形を変えるような場合は「編集モード」にまず切り替えます。

4.3.1 オブジェクトの追加

図 4.4 のように①の「追加」のメニューをクリックし、②の「メッシュ」のサブメニューをクリックすると追加するオブジェクトの一覧が出てくるので、「平面」～「モンキー」の中の追加したいものをクリックします。すると  (3D カーソル) の位置に新しいオブジェクトが追加されます。場合によっては大きすぎたり、小さすぎるオブジェクトが出てきます。追加後すぐであれば、画面の左下に「> ~を追加」というのがあるので、それをクリックすると大きさなどの設定のパネルが出てくるので数字を変更して直します。

3D カーソルの位置は、画面の左端にある図 4.5 のツールバー*16の②をクリックすると、マウスでドラッグしたり、画面をクリックすることにより変更することができます。

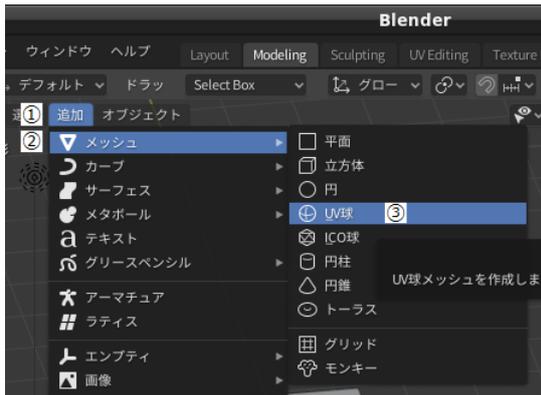


図 4.4 オブジェクトの追加



図 4.5 ツールバー

4.3.2 文字オブジェクトの追加

立体的な文字を作成することができます。なお、説明文のような文章は後述のテクスチャを利用して、文章の画像を貼り付けた方が良いでしょう。図 4.4 の②の下の方にある「テキスト」をクリックすると、3D カーソルのあった位置に「Text」と言う厚みのない文字オブジェクトが追加されます。Tabで「編集モード」に変更すると Back Spaceで文字を消したり、キー入力で文字を追加することができます。

文字に漢字などが含まれる場合は、フォントを変更しないと表示されません。「オブジェクトモード」で文字オブジェクトを選択して、右のパネルのところで、図 4.6 の①の下の「a」をクリックし、②の「フォント」のところを開き、③の下のフォルダーボタンをクリックするとフォントファイルの指定画面になります。ここで、「truetype」と言う名前のフォルダーアイコンをダブルクリックして開くとたくさんのフォルダーアイコンが表示されます。その最後の方に「日本語」のアイコンがあり「IPAexGotjoc Regular」とか「IPAexMincho Regular」と言う名前がついているので、ゴシック体が良ければ前者を、明朝体が良ければ後者をダブルクリックすると日本語が表示されるようになります。

図 4.6 の②の「フォント」の上の「ジオメトリ」を開いて、その中の「押出」に0より大きな値を設定すると厚みのある立体的な文字になります。文字間隔を変更する際には、図 4.6 の②の「フォント」の下の「段落」の中の「間隔」の中の「文字の間」で設定できます。

形状などの設定が終わったら、文字オブジェクトの形式をメッシュに変更します。これをしないと立方体などの他のオブジェクトとの結合などができませんし、3D プリンターで造形もできません。メッシュに直すには、文字オブジェクトを右クリックして出てきたメニューで、「変換」の中の「メッシュ」を選択します。するとこれまで固まりだった文字オブジェクトが平面で囲われた形になります。一度メッシュにすると元に戻すことはできません。フォントや文字の変更が必要となったら、もう一度文字オブジェクトの追加からします。

*16 Tを押すとツールバーの表示、非表示を切り替えることができます。



図 4.6 文字オブジェクトの設定

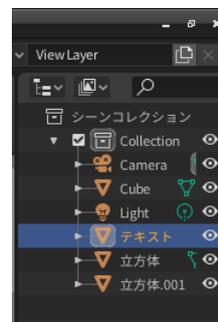


図 4.7 オブジェクトの名前の変更

4.3.3 オブジェクトの名前の設定

新しく追加されたオブジェクトには自動的に名前が付いています。機械的に立方体ならば「Cube001」のような名前が付くので、どんどん変形をして全く違う形になった場合識別する役に立ちません。オブジェクトをクリックして選択すると、画面の右上の図 4.7 のような「シーンコレクション」のところにあり、対応するオブジェクトのところが白くなるので、名前のところをダブルクリックして変更します。

4.3.4 オブジェクトの大きさや位置などの変更

移動：図 4.5 のツールバーの③をクリックすると、選択したオブジェクトに赤、緑、青の矢印が出てきます。これを使ってそのオブジェクトの原点を x 軸、y 軸、z 軸の方向に動かすことができます。青の矢印の向いている方向が z 軸なので上方向になります。この矢印をドラッグするとオブジェクトが動きます。矢印の根本にある黄色い点をドラッグすると自由な移動は可能ですが、立体的に正しい位置へ持って行くのはなかなか難しいと思います。ツールバーをクリックしなくても、**[G]**を押すと移動モードになります。マウスを動かすとそれによってオブジェクトが動きます。最後にクリックすると移動モードが終わります。

回転：図 4.5 のツールバーの④をクリックすると、選択したオブジェクトに赤、緑、青の円弧が表示されます。この円弧をドラッグするとオブジェクトを回転させることができます。ツールバーをクリックしなくても、**[R]**を押すと回転モードになります。マウスを動かすとそれによってオブジェクトが回転します。最後にクリックすると回転モードが終わります。

拡大・縮小：図 4.5 のツールバーの⑤をクリックすると、選択したオブジェクトに赤、緑、青の先が丸い矢印が出てきます。この矢印をドラッグするとオブジェクトの大きさが変わります。矢印をドラッグした場合、その方向だけオブジェクトの大きさが変わります。そのためにオブジェクトの形状は歪みます。形状を変えずに大きさを変更したい場合は、矢印の外側にある白い円をドラッグします。ツールバーをクリックしなくても、**[S]**を押すと拡大・縮小モードになります。マウスを動かすとそれによってオブジェクトの大きさが変わります。最後にクリックすると拡大・縮小モードが終わります。

4.3.5 数字でオブジェクトの大きさや位置などを設定

オブジェクトを 3D プリンターで実物にする場合、その大きさの設定をマウスで適当に済ますわけには行きません。数字できっちり設定するためにはまず単位を設定する必要があります。そしてオブジェクトを選択して**[N]**を押すと右側に設定用のパネルが出てきます。そこで位置、回転、寸法を数字で指定できます。「拡大・縮

小」のところは寸法を変更すると自動的に元の大きさに対する倍率が設定されます。逆にここで倍率を入力すると「寸法」のところの数字が自動的に設定されます。

なおこのパネルが不要になった場合は、もう一度 \square を押すと消すことができます。

4.3.6 オブジェクトの削除

オブジェクトを選択して、 \square を押すと、オブジェクトが削除されます。うっかり違うオブジェクトを削除してしまった場合は、 $\text{Ctrl}+\text{Z}$ で元に戻します。

4.3.7 オブジェクトの複製

オブジェクトを選択して、 $\text{Shift}+\text{D}$ でオブジェクトの複製ができます。できたオブジェクトは元のオブジェクトとぴったり重なっていますが、マウスをそのまま動かすと出てきます。適当な位置まで動かしてクリックすると位置が確定します。

4.3.8 オブジェクトの統合

Shift キーを押しながらクリックすると、複数のオブジェクトを選択することができます。そして $\text{Ctrl}+\text{J}$ で選択した複数のオブジェクトを一つのオブジェクトにすることができます。なお、この方法で統合したオブジェクトは、編集モードで P でメニューを出し、「構造的に分離したパーツで」を選択して、元のオブジェクトに分けることができます。

4.3.9 オブジェクトの計算

オブジェクトに丸い穴を開けたい場合、オブジェクトを後述の変形操作で丸く削るのは困難です。一方丸い穴の形状は通常円柱で、この形のオブジェクトは簡単に作ることができます。よって元のオブジェクトから円筒オブジェクトの部分を引き算すれば、丸い穴の空いたオブジェクトを簡単に作ることができます。このようにあるオブジェクトに、別のオブジェクトを加える(結合)、切り取る(差分)、重なり部分だけにする(交差)ことができます。

まず元のオブジェクトを選択し、右のパネルのところで、図 4.6 の①の 2 つ上の  (モディファイア) をクリックすると、 \square が出てくるのでクリックすると、図 4.8 の右のようなメニューが出てくるので、「生成」をクリックするとさらにサブメニューが出てくるので「ブーリアン」を選択すると図 4.9 のようになります。ここで、「差分」、「結合」、「交差」を選択し、「オブ...」の右にある \blacksquare をクリックするともう一つのオブジェクトが選択できます。最後に図 4.9 の一番上のカメラのアイコンの隣の \square をクリックするとメニューが出てくるので「適用」を選択すると実行されます。結果は元のオブジェクトと重なっているのに、移動させないと結果が見えないと思います。

4.4 オブジェクトの変形などの操作

Tab で「オブジェクトモード」と「編集モード」の切り替えができます。「オブジェクトモード」ではオブジェクトの位置などのオブジェクト全体の操作ができましたが、「編集モード」ではオブジェクトの部分に対する操作ができます。オブジェクトを編集モードにすると、画面左上が図 4.10 のようになり、「編集モード」の横の 3 つのボタンで編集対象を点、線、面に切り替えることができます。図 4.10 は一番左が青くなっているので、点が編集対象になっています。ここでオブジェクトの適当な点をクリックして選択し、その点に対してオブジェクトの移動と同じ操作を行うと、選択した点のみが動くので、オブジェクトの形が変わる事になります。

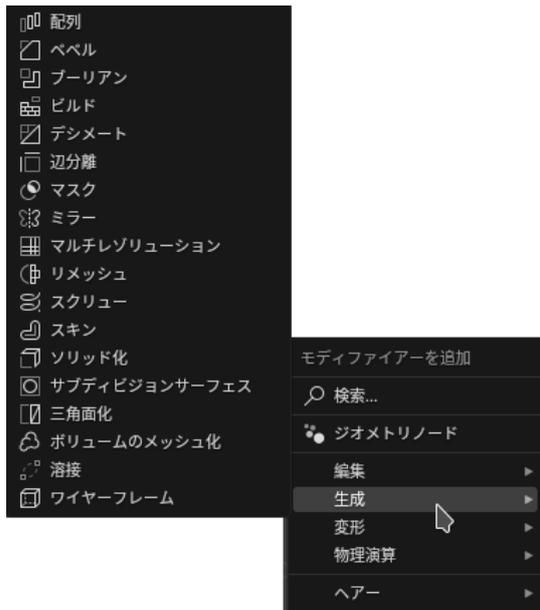


図 4.8 モディファイアのメニュー



図 4.9 ブーリアンの設定



図 4.10 編集対象の選択

4.4.1 オブジェクトの原点を底面に移動

3D プリンターで出力する 3 次元モデルは宙に浮いては困ります。またオブジェクト同士の位置をきちり数字で設定するような場合、オブジェクトの原点がわかりやすい位置にないと面倒です。オブジェクトの底面となる面を選択し、**(Shift)+[S]**で円環状のメニューが出るので「カーソル→選択物」を選択して 3D カーソルを面の中央に移動します。そして、**(Tab)**で「オブジェクトモード」に変更して、図 4.11 のように画面の左上の「オブジェクト」の横の方にある「オブジェクト」メニューの中の「原点を設定」をクリックし、「原点を 3D カーソルへ移動」を選択します。この場合原点が 3D カーソルの方にやってくるので、見た目は変わりません。**(N)**を押して位置の数字を全てゼロにすると、底面が床? に着いて正しく設定できたかどうかわかります。

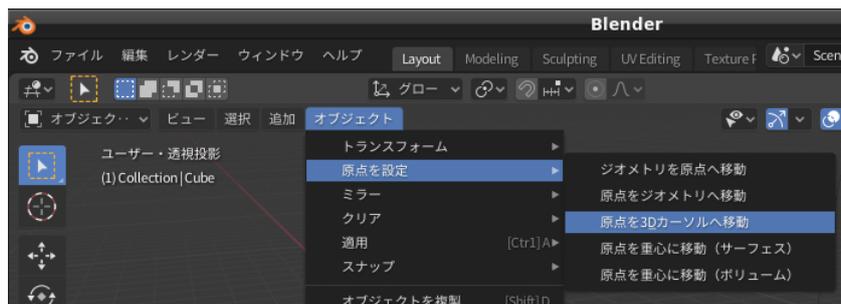


図 4.11 オブジェクトの原点の設定

4.4.2 面の細分化

元となるオブジェクトが少ない数の面で構成されていると、形を細かく設定することができません。そのような場合は面を細分化するとやりやすくなります。オブジェクトの細分化したい面をクリックして選択し、右クリックでメニューを出して「細分化」を選択します。この場合四角形に分割されますが、メニューで「扇状に分割」にすれば扇形に分割することもできます。

4.4.3 角を丸める

我々が手にするものは大抵角が丸く作られています。そうしないと手などに当たった時に怪我をする恐れがあるからです。オブジェクトの角を丸くするには、面や辺を選択した上で、**[Ctrl]+[B]**で「ベベル」と言う面取りの状態になります。マウスを動かすとどのくらいの幅を丸くするか(量)が変わり、ホイールを動かすと分割数が変わります。適当なところでクリックすると確定します。すると画面の左下に「ベベル」というものが現れるのでクリックするとパネルが開きます。パネルの一番上の「オフセット」の数字で元の角を押し込む幅、「セグメント数」の数字で分割数を設定することができます。

4.4.4 テクスチャの貼り付け

オブジェクトの表面に画像を貼り付けることによりオブジェクトの質感(ザラザラしているとか)や模様を付けることができます。この貼り付けられた画像のことをテクスチャと呼びます。ただ画像は平面ですが、オブジェクトの表面は多角形です。よって通常のオブジェクトにテクスチャを設定するには、そのオブジェクトの展開図を作成し、それに合った画像を用意するような手順となります。画像の作成はBlenderで行うこともできますし、一旦別のソフトで編集したものを取り込むことも可能です。なおテクスチャはオブジェクトに直接結びつくのではなく、マテリアルに付く形になるので、もしオブジェクトにマテリアルが設定されていない場合は、まずマテリアルを設定する必要があります。

4.5 オブジェクトなどの保存

三次元モデルが完成したら、画面最上部にある「ファイル」メニューの中の「名前を付けて保存」を選択します。そして保存場所やファイル名を設定して保存します。拡張子は「.blend」が付きます。これはシーン全体の保存となりオブジェクトだけでなく照明やカメラなども含まれます。

4.6 プレートの作成

個々の操作の説明だけでは、なかなか実際のオブジェクトの作成方法は分かりません。ここでは図4.12のような名前プレートの作成を例にして、Blenderでいかに三次元モデルを作成するかを説明します。なお()の間の数字はやり方を説明している章の番号です。

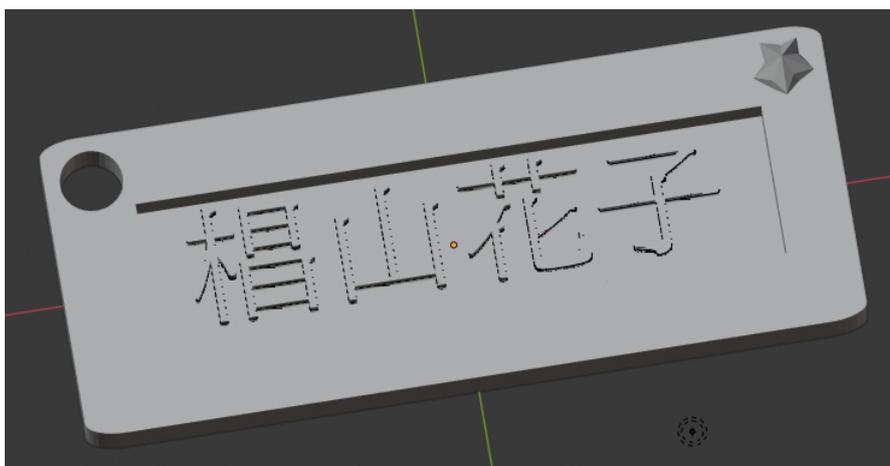


図 4.12 名前プレート

1. Blender を起動して、単位を mm に設定する。(4.2.1)
2. 最初からある立方体 (Cube) の大きさを 50mm×20mm×3mm にする。(各々 xyz の値)(4.3.5)

3. Cube の下の面の中央を原点にする。そして Z 座標をゼロにして Cube を平面上にする。(4.4.1)
4. Cube の 4 つの角を丸くする。オフセットは 0.1mm、セグメントは 8 にする。(多少異なる値でも構わないが 4 つの角が全て同じ丸みになるようにする)(4.4.3)
5. 立方体を追加しその大きさを 40mm×10mm×4mm にする。(4.3.1)
6. 立方体を動かして Cube に重ねる。立方体の位置を $x=0.5\text{mm}$ 、 $y=0$ 、 $z=3\text{mm}$ にする。(4.3.4)
7. Cube から立方体と重なった部分を切り取る。(差分) そして立方体をずらしてちゃんと凹みがついていれば、立方体を削除する。(4.3.9)
8. 半径 2mm、長さ (深度)10mm 程度の円柱を追加する。これを Cube の左上の部分に重ねた ($x=-22\text{mm}$ 、 $y=7\text{mm}$) 上で、重なった部分を切り取る。うまく Cube に丸穴が空いたら円柱を削除する。(4.3.9)
9. 星形を付ける。半径 2mm、長さ (深度)1mm、頂点数 10 の円柱を追加する。円柱の上面の頂点を一つおきに内側に動かし星形にする。さらに上面を扇形に分割し、中心点を少し持ち上げて立体的な星形にする。これを Cube に埋め込んで結合させる。(4.4.2)
10. 文字オブジェクト (楢山花子) を追加する。押し出しを 2mm にする。また文字の大きさは Cube の凹みに入り、丸穴や星形と重ならない程度にする。メッシュ形式に変換した上で $z=2.5\text{mm}$ にして Cube に埋め込み、結合し、文字オブジェクトを削除する。(4.3.2)
11. 名前を付けて保存する。(4.5)

5. 3D プリンター

通常のプリンターは平面の紙の上に字や絵を書きます。3D プリンターは立体を作ることができます。様々な方式のプリンターがあり、立体を作成するために使用する物質によってもできるものが変わります。この授業で使用する 3D プリンターは、FFF 方式^{*17}と呼ばれる比較的低い温度で柔らかくなるプラスチックを利用して、紐状の粘土で土器を作るように立体を造形します。色はプラスチックの色になりますので、通常は単色の立体になります。高級な 3D プリンターでは途中で別の色のプラスチックに切り替えることによって別の色の部分を作ることができますが、何種類もの色というわけには行きません。しかし、比較的構造が簡単で、プラスチックもそれほど高価ではないため、個人的な利用や立体の試作などに使われています。

用途によってはフィギュアのように色がないと色気がない場合もあります。そのような場合は石膏を利用したものが用いられます。石膏の粉に色付きののりを吹き付けて造形します。インクジェットのプリンターのインクにのりが混ざっている感じで、平面が一つ分描いたら、その上に石膏の粉を薄くまいてまた描くという繰り返しで造形します。最終的には石膏の中からのりで固まった立体を掘り出し、さらに石膏は柔らかいので表面を薄くコーティングします。これも石膏が比較的安いので安価ですが、プリンターの大きさはかなり大きくなります。

工業的に利用するものは、プラスチックや石膏では強度が足りない場合も多いので、金属による造形が可能な 3D プリンターが用いられます。チタンによるものは、石膏と同様な方式で造形が可能なためチタンそのものは安くありませんが、一番安価に手に入るようです。ただしチタンの粉を固めるためにはのりでなく、レーザー光線で熱するため色は付けられません。それ以外の金属として、鉄、アルミニウム、銀、白金、金なども可能ですが、これらは一旦石膏などで型を作成し、そこへ溶けた金属を流し込んで作成するという手間のかかる方法を使用するため、時間や費用がかかります。

5.1 3D プリンターの使い方

mars には「Flash Forge Adventure III」という比較的低温で柔らかくなるプラスチックを利用した 3D プリンターが接続されており、三次元モデルさえあれば、簡単に 15cm×15cm×15cm より小さい立体を造形することができます。材質は ABS(アクリロニトリル・ブタジエン・スチレン) と PLA(ポリ乳酸) が使用可能で、現在は白色の PLA がセットされています。PLA は ABS よりも低温で柔らかくなり、固くて脆いので加工しにくい、塗料が付きにくいというような特徴があります。逆に言えば ABS の方が高温に耐えて、柔軟性があるので落としても割れにくいものことができます。以下に造形の手順を示します。

1. Blender で三次元モデルを作成し、上部の「ファイル」メニュー→「エクスポート」→「Stl (.stl)」と選択し、STL と呼ばれる形式で保存します。
2. 画面左下の「スタート」→「アクセサリ」→「FlashForge 3D Printer Control Software v5」(以下プリンター制御ソフトと呼ぶ) と選択します。
3. 「ファイル」メニューの「ロードファイル...」で STL 形式で保存したファイルを読み込みます。すると図 5.1 のようになります。
4. 「編集」メニューの「サポート材」で空中に浮く部分の足場を生成します。図 5.2 のような設定画面で **自動サポート** をクリックし、サポート材が生成されたら **もどる** をクリックします。もし「プレートと接続する」のチェックがついていたら外してください。
5. 画面の上部に出てくる **スライス実行** ボタンで立体モデルを輪切りにします。図 5.3 のような設定画面が出ます。「標準」、「細かい」、「速い」のどれかをクリックします。「細かく」にするとより綺麗な造形が可能ですが時間がかかります。「速い」にすると短い時間でできますが、荒いものになります。 **スライス**

^{*17} Fused Deposition Fabrication、FDM 方式とも呼ばれる。

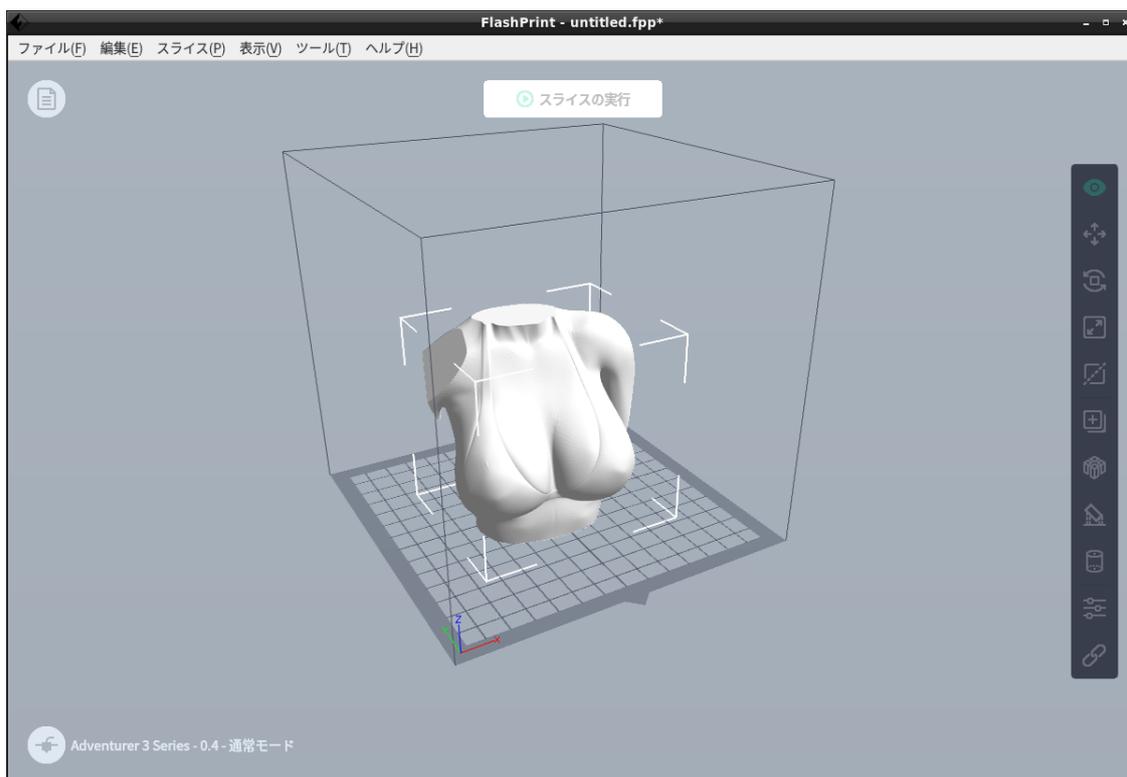


図 5.1 三次元モデルを読み込んだ状態



図 5.2 サポート材設定画面



図 5.3 スライス設定画面

ボタンをクリックすると輪切りを実行します。上部に「スライスのプレビュー」が表示されるのでこれをクリックするとどのくらいの樹脂を使用するかや出力にかかる時間を見ることができます。

6. 「スライス」メニューの「プリンターに接続する」を選択すると、接続先を尋ねてくるので、「自動スキャン」でプリンターを表示させて「全て接続」をクリックします。すると「マルチ制御機器」の画面に代わるので、プリンターのアイコンの前のチェックをクリックして、「Gコードを送信」ボタン^{*18}をクリックします。すると「Gコードファイル名」を尋ねてくるので通常そのまま「OK」をクリックします。
7. 印刷が開始されると、ヘッドなどの温度や進み具合が表示されます。プリンターの動いている様子を、図 5.4 のように <http://mars.mgt.sugiyama-u.ac.jp/3dp.htm> で見ることができます。このページはパソコンやスマホからでも見えます。



図 5.4 3D プリンターのモニタ画面

8. 印刷が終了したら、4 階の 414 室に行って造形された立体モデルを取り外します。取り外さないと次の人が出力できないので、終了したらすぐに取り外してください。立体モデルの乗っている黒いプレートの手前の部分を少し下に押しと手前にプレートを引き出すことができます。プレートを少しひねるとパリと言う感じで立体モデルが剥がれます。

複数の人が同時に 3D プリンターを使用した場合、まぜこぜの立体モデルができる事はありません。最初の人が使用中は、次の人は 3D プリンターに接続できません。造形が終わると次の人が 3D プリンターに接続できるようになります。そこで最初の人が造形された立体モデルを取り外していない場合、3D プリンターヘッドが衝突してモデルか 3D プリンターが壊れることとなります。よって印刷開始の前には、必ず 3D プリンターの内部を確認してください。

^{*18} G コードは元々工場で使われる工作機械の制御に使われるものです。工作機械では金属の塊から作りたいものを削り出して作りますが、3D プリンターでは逆に積み重ねていく感じで作るので、ヘッドの動きなどが外側からでなく内側からと言うように逆になります。

5.2 3D プリンター使用上のヒント

インターネットで検索すればたくさんの STL 形式のファイルを見つけることが可能です。これらを頂いてきて、3D プリンターにかけることも可能ですが、大きさがプリンターの造形可能なサイズを超えている事もあります。そういう場合は、ファイルを読み込んだあとで、画面の右側にある縦に並んだボタンの上から 4 番目の「スケール」ボタンで大きさを変更することができます。

この授業で使用する 3D プリンターは、立体モデルの下の方から造形します。途中では宙に浮いてしまう部分があるとサポート材による支えが必要になります。全て造形が済めば宙に浮いた部分もつながるので、サポート材を物理的に切り離して完成となります。複雑な立体モデルでは内部にこのサポート材が必要となり、後でそれを除去できなくなるという事もよくあります。支柱の部分に使われたプラスチックは無駄になりますし、除去の手間を考えるとできるだけサポート材は少ない方が良いでしょう。例えば蓋のない箱の場合、蓋のない面を上に向ければサポート材は不要ですが、蓋のない面を下にすると箱の中はサポート材だらけになります。この辺りを考慮して立体モデルの作成の際に向きを決めましょう。

画面の右側にある縦に並んだボタンの上から 5 番目の「カット」を利用して立体モデルを分割することができます。例えば球はどのような向きにしても、床面と点で接触するのでサポート材が必要になります。しかし半分に切り分けて断面を下にすれば、単なる丸い山形ですからサポート材は不要です。造形した後で貼り合わせる必要が生じますが、サポート材を外すのよりは楽でしょう。

6. 拡張現実

仮想現実 (VR: Virtual Reality) というものがあります。コンピュータが作り出した仮想世界をゴーグルの形をしたディスプレイ (HMD: Head Mounted Display) で見るもので、頭や体の動きに合わせて見せる仮想世界を動かすので、本当にその世界の中に居るような感じになります。コンピュータの性能向上のおかげで本当に綺麗な仮想世界を見せることができるようになりました。一方問題点は次のとおりです。

- 綺麗な仮想世界を作るためには膨大な三次元データが必要となる。見渡す限りに存在する全ての物を用意する必要があります。全て空想の世界であれば、単純な立体で構成することも可能でしょうが、現実に近い世界を作るとなると細かいところまで忠実な三次元データを作成しなければなりません。
- 頭や体の動きと仮想世界の動きにはどうしても遅れやズレがあり、その結果長時間使用すると車酔いのような症状をもたらすことがあります。

これに対して拡張現実 (AR: Argumented Reality) は現実世界の画像の中に、コンピュータが作り出した仮想物体を表示するものです。例えば「ポケモン GO」では、スマホのカメラで写し出した現実の風景の中に、コンピュータの作り出したポケモンが登場します。周りのものや背景は全て現実のものを借りるので、仮想物体だけ用意すれば済みますので VR よりも楽です。また VR のように頭や体の動きの検出の必要もありません。もちろん全く現実と異なる世界を提供することはできません。現実世界のどこに仮想物体を出現させるかによって AR は大きく 2つの方式に分かれます。

- **マーカーと呼ばれる図のあるところに出す方式**：マーカーのあるところを基準にして、大抵その上に仮想物体が居るように表示されます。マーカーを動かしたり、マーカーに近寄るとその動きに合わせて仮想物体が動いたり大きさが変わったりします。マーカーが必要ですが、マーカーさえあればどこにでも仮想物体を出すことができます。現在はマーカーとして普通の絵も利用可能です。例えば「日経 AR」は日経新聞記事中のグラフや写真をマーカーとしています。
- **GPS などで求めた位置を基準に仮想物体を出す方式**：「ポケモン GO」がこの方式で、おかげでポケモンを探してあちこち歩く必要があります。現在の GPS では位置の精度には数 m の誤差がありますので、出現する位置もその影響を受けます。また屋内などの GPS の電波が十分届かない場所では使えません。GPS に頼らず、スマホの向きとカメラに写った内容から地面を検出して位置を特定して仮想物体を出すことも可能になりました。こちらはスマホからの相対的な位置にマーカー無しで仮想物体を出すことができます。

両者を活かしたものもあります。2020 年 10 月に発売された「マリオカートライブホームサーキット」では無線操縦で模型のマリオカートを室内で走らせます。マリオカートにはカメラが付いていて、ディスプレイでマリオカートの前方の風景を見ることができます。ディスプレイには現実の室内にある家具などに加えて、競争相手や障害物、床の上にはコースのラインなどが表示されます。さらに組み立て式のゴールなどがマーカーになっています。

この演習では大量の三次元データや HMD を必要としない AR を取り上げます。またマーカーさえあればどこでも使える方式を採用します。作成した三次元モデルを 3D プリンターを使えば現実の物にすることができましたが、AR を使えば三次元モデルが現実にあるように見せることができるようになります。

6.1 AR テストシステムの使い方

スマホのアプリで AR を検索すると多数のアプリが出てきます。それぞれ出てくる仮想物体が違います。要するに表示される物の種類だけアプリがあります。Blender で自分が作成した三次元モデルを出せるようなスマホアプリを作りかけたのですが、間に合わなかった (できなかった?) のでブラウザで見るシステムを借りま

した。以下のような手順で使います。なお、単純な図形の表示で良ければ obj 形式の三次元モデルなしでも使えます。

1. Blender で三次元モデルを作成する。

この章で使用する AR システムでは、Blender で単位をメートルにして、三次元モデルを作成することを想定しています。また原点がマーカーの中心部となっているので、原点はモデルの背面の少し後ろに設定すると、モデルがマーカーの上に浮いて見えるようになります。

最初からメートル単位を設定してモデルを作成するのではなく、3D プリンターでの出力を想定して単位を mm にして作成したモデルを使用するならば、次のようにして大きさを修正してください。

- a. 「シーン」の設定で、「単位の倍率」を「0.001」から「1」に変更する。
- b. 同じく、「長さ」を「ミリメートル」から「メートル」に変更する。
- c. AR で見せるオブジェクトをクリックして選択し、**N**を押して、「スケール」を表示させて、この3つの数値を全て 1/1000 のものに書き換える。

2. 三次元モデルを obj 形式で保存する。

- a. 対象をクリックして選択する。
- b. 「ファイル」メニューの中の「エクスポート」の中の「Wave front (.obj)」を選択する。
- c. 保存場所とファイル名を指定してから**OBJ をエクスポート**をクリックする。これで指定した「～.obj」ファイルと同じところに「～.mtl」ファイルが保存される。

3. AR テストシステムへブラウザからアクセスする

「<https://mars.mgt.sugiyama-u.ac.jp/AR/>」へアクセスすると図 6.1 のように表示されます。



図 6.1 AR テストシステムの最初の画面

4. 「新規登録」をクリックすると図 6.2 のように表示されますので、「タイトル」、「パスワード」を入力し、「マーカーの種類」は適当に選択し、「マーカーの大きさ」は作成したモデルと近い大きさを入力します。「ターゲットの種類」は必ず「Blender などで作成した obj ファイル (要アップロード)」を選択します。さらにその下で、「ターゲットの 3D モデルの OBJ ファイル」と「ターゲットの 3D モデルの MTL ファイル」を指定します。全て入力し、選択し、指定できたら「新規登録」をクリックします。
5. 「新規登録」をクリックすると図 6.3 のように URL や QR コードが表示されるので、これらをもとにスマホやタブレットでアクセスします。
6. タブレットやスマホでアクセスすると、背面カメラによる映像が表示されるので、カメラをマーカーの方に向けると図 6.4 のように三次元モデルが出現します。映像が表示される前にカメラの使用の許可を求めてきた場合は、毎回許可しないと表示されません。

ARデータの登録

マーカーやそれに対して表示するものを指定してください。パスワードは登録内容の修正や削除の際に必要になりますので、何を入力したか忘れないでください。入力が終わったら「新規登録」をクリックしてください。 [新規登録せずに戻る](#)

ページタイトル: (ARのWebページのタイトル)

パスワード: (修正や削除の際のパスワード)

マーカー: マーカーの種類: **hiro** マーカーの大きさ: cm
 マーカーの画像ファイル: [参照...](#) ファイルが選択されていません。

ターゲット: ターゲットの種類: **色付き箱**
 A-Frameのコマンド: (ターゲットがobjファイル指定の場合は追加設定など [A-Frameコマンド例](#))

OBJファイル: [参照...](#) ファイルが選択されていません。 MTLファイル: [参照...](#) ファイルが選択されていません。
 IMGファイル: [参照...](#) ファイルが選択されていません。(必要があれば)

[新規登録](#)

by K.Miki 2019/09/03, 09/13, 09/17, 2020/11/15

図 6.2 新規登録の画面

新規登録結果

| | |
|-------|--|
| タイトル | sugiyamatest |
| パスワード | 9999 |
| URL | https://mars.mgt.sugiyama-u.ac.jp/AR/s.php?n=386  |
| マーカー | hiro マーカー表示 |

[もどる](#) by K.Miki 2019/09/03, 09/05, 09/13, 10/25, 2020/11/15

図 6.3 登録結果の画面

6.2 ブラウザによる AR システム

予め AR アプリをインストールする必要のないブラウザによる AR システムは、手軽な AR システムとしてもっと普及しても良いのではないかと思います。ただ全ての処理を JavaScriptで行っており、マーカーの検出などの処理に不安を感じる他、毎回 Web サーバーから全てを受け取らなければならないため、時間がかかったり、ネットワークのないところでは使えなかったりします。

前節で使い方を説明した AR テストシステムは、Jerome Etienne 氏の 2017 年 7 月 11 日のブログの記事「Creating Augmented Reality with AR.js and A-Frame」^{*19}をもとにしたものです。「A-Frame」は仮想現実の世界を HTML のようなタグで記述し、ブラウザでそれを見ることができるようになろうとする試みです。例えば次のような記述のみで、水色の立方体、赤い球、黄色の円筒が薄い水色の面の上にあるように表示され

^{*19} <https://aframe.io/blog/arjs/>

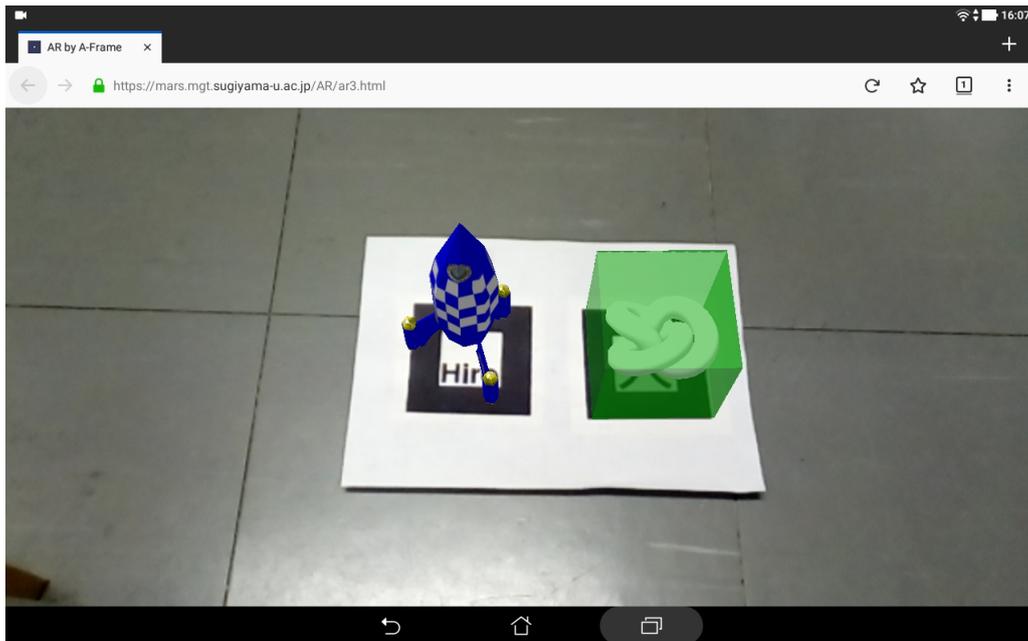


図 6.4 タブレットでの表示

るだけでなく、マウスでドラッグすると見る方向を変えることもできます*20。

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D">
      </a-cylinder>
      <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4"
        color="#7BC8A4"></a-plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

JavaScript を使用して独自のタグが使えるようにしています。そしてこれで記述した三次元モデルが、カメラで撮影したマーカーの位置に表示されるようにするのが AR.js です。こちらでも独自のタグでマーカーの種類などを指定します。以下はこれらの独自のタグの使い方の一部を紹介します。

まず単独の赤い箱です。

```
<a-box position='0 0.5 0' material='opacity: 0.5; side: double; color: red;'>
</a-box>
```

position で箱の位置を指定し、material ではこの材質を指定しています。opacity は不透明度で数字が大きくなるほど中は見えなくなります。

次は青い箱の中で紐がぐるぐるするものです。

*20 <https://mars.mgt.sugiyama-u.ac.jp/AR/aframe.htm> へアクセスすると見ることができます。

```
<a-box position='0 0.5 0' material='opacity: 0.5; side: double;color: blue;'>
  <a-torus-knot radius='0.26' radius-tubular='0.05'
    animation='property: rotation; to: 360 0 0; dur: 5000;
    easing: linear; loop: true'>
  </a-torus-knot>
</a-box>
```

a-box のタグの間に紐のタグである a-torus-knot が入っています。radius は半径のことでこれで紐の大きさを設定します。animation で動きの設定をします。property で動き方を回転 (rotation) としています。to でどこまで回転するかを指定しています。三次元なので回転軸も3つあるので数字が3つ並んでいます。dur は回転周期ですので、小さな値にすると早く回転します。loop が true になっているのでずっと回転します。

OBJ 形式の三次元のモデルを動かす場合は、a-obj-model のタグに animation の指定を追加します。

6.3 テクスチャ付きモデルの作成方法

テクスチャとはモデルの表面の模様のことです。本来は3章で説明すべきことですが、Blender でテクスチャ付きモデルの作成方法をここで説明します。現在本学部にある3Dプリンターでは色付きのモデルを造形することができませんが、AR では簡単に表示することができます。むしろ色ぐらい付いていないと寂しいのと、立体モデルとしては簡単な形状だがテクスチャによってはARで使えるものも考えられるからです。

モデルの材質を示すマテリアルには色の設定があるため、これを利用すれば色付きモデルを作成することができます。ところが同じマテリアルの部分は同じ色になるので細かい模様のようなものは表現できません。そこで色付きのモデルでは大抵マテリアルの上に模様であるテクスチャを貼り付けています。大抵のフィギュアの顔の部分も目玉などはテクスチャで描いています。テクスチャのデータとなる画像は普通の平面画像です。これを立体に貼り付けるために結構面倒な手順が必要になります。特に顔のような凹凸が色々ある場合、展開図を作成してそこに色を付けるような形になります。ここでは一番簡単な平面にテクスチャを貼り付ける方法を説明します。

1. 貼り付ける画像を用意する。画像はPNG形式のものとし、後でOBJ形式のファイルを保存するところ、例えばデスクトップに置く。
2. Blender を起動し、初期設定の立方体を削除する。単位はメートルでなければメートルに変更する。
3. 22ページの図4.4のようにメニューを開き、最後に「UV球」でなく「平面」をクリックして平面を追加する。
4. Nを押して寸法設定画面を呼び出して大きさを設定する。画像の元となったものと同じ大きさにする。それが無理な場合は、せめて縦横比は画像と同じにしないとテクスチャが歪む。
5. 右側のところで図6.5の①をクリックしてマテリアルの設定を出して、②の「新規」をクリックしてマテリアルを追加する。
6. マテリアルを追加するとさらに様々な設定ができるようになる。図6.6の赤の矢印の先にある「ベースカラー」の○をクリックするとメニューが表示されるのでその中の「画像テクスチャ」を選択する。するとフォルダーの形をしたボタンが現れるので、それをクリックして画像ファイルを指定する。
7. 画面右上の図6.7のような表示のメニューを、赤の矢印の先にある「マテリアルプレビューモード」をクリックして変更すると、貼り付けられた画像の様子が見えるようになる。
8. 「ファイル」メニューの中の「エクスポート」でOBJ形式で保存する。保存先は貼り付けた画像があるディレクトリ(フォルダ)にする。
9. ARシステムに登録する際には、OBJファイル、MTLファイル、画像ファイルの3つを必ず指定する。

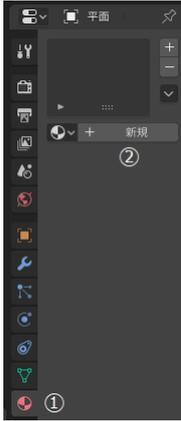


図 6.5 マテリアルの追加



図 6.6 マテリアルに画像を設定

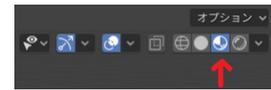


図 6.7 マテリアルプレビューモード

6.4 マーカー例

以下はマーカーの例です。



図 6.8 4cm 角 (内側は 2cm 角)

演習問題

以下のようなものを AR で実現せよ。ただし誰が登録したものかわかるように、タイトルに自分の名前か id を追加すること。

1. 黄色の箱の中で青い箱がぐるぐる回るもの
2. 前章で作成した名前プレートが、ゆっくり回転して裏も見えるようなもの
3. 有名絵画を実物大で見ることができるもの



図 6.9 8cm 角 (内側は 4cm 角)



図 6.10 12cm 角 (内側は 6cm 角)

7. ChatGPT

2022年11月にアメリカのOpenAIが公開したAIチャットボットのChatGPTは大きな話題になり、その活用が世界中で進められています。ChatGPTは大きく分けると2つの部分から構成されています。ChatとGPT (Generative Pretrained Transformer) です。GPTは大規模言語モデルの一つで、インターネットなどで収集した大量の文章をもとに、ある単語のまとまりの次に来る単語を学習しています。これによって問いかけに対する答えが返るシステムができそうですが、実際はそうは行きません。大量に例文を学ぶほど、様々な続きの可能性を学ぶことになり、答えを絞ることができなくなります。例えば、OpenAIが公開しているGPTの部分だけのモデルである「davinci」に「今日の天気は」と問いかけると次のような続きが返ってきました。(以下ChatGPTからの応答は四角い枠に入れてます。)

[Photos provided by NIGATA Shimbun.]

ミ Name ミ ミ Name ミ 1. Name ミ ___) 勺 (勺_____ 2. Name ミ e|||| ペロペロ ベチベチ 3. Name ミ ミ (・_| \| \ i||||i ペシ 4. Name ミ ミ [三] i||||iii||| さて (;e) ペタ? 5. Name ミ _____ |||||||||||| 穂わらしていただけた日はお日様のように明るいよ (^_-) ニヒ-|||{ ~ (*-☆ ジャネ… 6. Name ミ ,; ; ベ--|_ 《 (・D・へ) ■ポタツ 7. Name ミ (_ ∩ ∩) 8. Name ミ (_ _ "US イラルチ [" ~ィへハ US イラルチ 9. Name ミ ||||||||||||||||||||||||| [ニトツサネ・・・ | |時|ミえ 10. Name ミ ||||||||||||||||||||||||||||||||||||| r、()) () Vシ| |亡| 11. Name ミ ||||||||||||||||||||||||| トr|_ノ;

シ(°)(°)「おーい、おーい 大将!! 今度は【意味不明版】に2匹いる!! よーし…思い出さないなあ!!」(ハード 凹(ヨシュル) ※天気予報: こんな時は曇りで湿気も少なく涼しい中で割と暑い日

最初の例は画像ファイルのようです。後の2つの例の半角文字は都合により全角文字に直していますが、意味不明です。これに対して問とその答えのやり取りを追加学習 (fine-tuning) させます。ChatGPTで具体的にどのようなやり取りをどのくらいGPTに学習させたのかは分かりませんが、開発元のOpenAIによると、質問に文章で回答する形ならば500例以上、質問にYes/Noのような単語で答える形ならば各単語について100例以上学習させると良いそうです^{*21}。

「つくよみちゃん会話 AI 育成計画 (会話テキストデータセット配布)^{*22}」では、マルチクリエイター・夢前黎 (Rei Yumesaki) がプロデュースするフリー素材キャラクター「つくよみちゃん」の会話テキストを公開しています。2023年6月26日にダウンロードしたデータには表3のような469例がありました。これを「davinci」に追加学習させて「今日の天気は」と問いかけると次のような続きが返ってきました。

天気は晴れです!

晴れ用気になりますか? 雨用気になりますか?

ピクピクしていますね!

よく分からない事も答えてますが、追加学習なしのものの回答と比べるとともです。追加学習データにあった「いい天気だね」を問いかけると、

^{*21} Fine-tuning - OpenAI API: <https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset>

^{*22} <https://tyc.rei-yumesaki.net/material/kaiwa-ai/>

| |
|-------------------|
| 雨が心配な方はお気を付けください。 |
| そうですね。晴れ男 |
| それはよかったね！ |

のように返ってきました。教えた通りの答えはなかなか返ってきません。一方回答を見ていると、なんとなく回答者の雰囲気が感じられます。500 例ほどで回答者の振りができると言うのはなぜでしょうね。

表 3 会話テキストデータの最初の 5 件と「天気」が出てくる 2 件

| 質問 | 答え |
|-------------|---------------------------------------|
| お腹が鳴る | 何か召し上がりますか？ |
| だるい | それは心配です。私にできることがあれば、何でもお申し付けください。 |
| 眠れない | 何も心配する必要はありません。私がついています。ずっと、ずっと、一緒です。 |
| 押すなよ、絶対押すなよ | はい、絶対押しません！ |
| 見せられないよ | 目をつぶって待っています。 |
| いい天気だね | それはよかったです！ 月は見えますか？ |
| 明日の天気は？ | WEB 検索してみたいかがでしょうか。 |

7.1 ゲームプログラムを作る

昔コンピュータが初めて登場した時から、コンピュータを利用するにはプログラムを作らなければならない、という問題がありました。最初は利用者がプログラムを作っていました。やがてコンピュータの利用が広がるに従い、プログラムを作る人と使う人が分かれてきました。Excel などの表計算ソフトがなかった頃は、簡単な計算もプログラムの作成が必要でした。表計算ソフトが利用できるようになり、簡単な計算を行うことは容易になりましたが、複雑な計算をさせるためには表計算ソフトの使い方を学ぶ必要があります。最近日本では小学生からプログラミングを学ばせるようになりましたが、表計算ソフトも十分使いこなせない人が多いことから、プログラミング嫌いを大量生産するだけではないかと心配しています。一方 ChatGPT の助けがあれば、誰でもプログラムを作ることができるようになるかもしれません。

ここでは簡単なゲームプログラムを ChatGPT を利用して作成する例を紹介します。まず「gpt-4o」というバージョンの ChatGPT のを使用した例です。角の丸い四角で囲まれた部分がこちらが入力した内容で、普通の四角で囲まれた部分が ChatGPT からの応答です。

質問: JavaScript を使ってシューティングゲームを作りたいと思います。まずキー入力で左右に動かせる青い自機を表示し、キー入力で緑色の弾を撃てるようにしてください。なおプログラムは一つのファイルにしてください。(71 tokens)

回答: JavaScript で基本的なシューティングゲームを作成するためのサンプルコードを以下に示します。これは、HTML5 の「<canvas>」を使用して青い自機を左右に動かし、緑色の弾を発射するという基本的な機能を含んでいます。

```
html
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>シューティングゲーム</title>
```

```

<style>
  canvas {
    background-color: #000;
    display: block;
    margin: 0 auto;
  }
</style>
</head>
<body>
  <canvas id="gameCanvas" width="800" height="600"></canvas>
  <script>
    . . . 中略 . . .
  </script>
</body>
</html>

```

このコードの説明

- ****キャンバスの設定****: ‘<canvas>’タグを使用してゲームの描画領域を作成しています。
- ****自機の設定****: 青い四角として描画される自機の初期位置とサイズ、速度を定義しています。
- ****弾の設定****: 緑色の弾は配列として管理され、発射されるたびに新しい弾が追加されます。
- ****キー入力処理****: ‘keydown’ と ‘keyup’ イベントを使ってユーザーのキー操作を検知し、左右移動とスペースキーによる弾の発射を実現しています。
- ****ゲームの更新と描画****: ‘update’関数でオブジェクトの状態を更新し、‘draw’関数でキャンバスに描画しています。‘requestAnimationFrame’を使ってゲームループを実行します。

この基本構造をもとに、敵キャラクターや衝突判定、スコアシステムなどを追加していくことで、より複雑なシューティングゲームを作成することができます。(920 tokens)

これで中略部分を含めて 94 行のプログラムです。僅かな時間で出力されました。

質問や回答の最後に「920 tokens」のように token の数が付いています。ChatGPT の内部では文字ではなく、ほぼ単語と同等である token という形で処理を行っています。ChatGPT の使用料はこの token の数に比例して掛かります。数字を見ると回答の token の数は 900~1,500 のように徐々に増えているのに対して、質問の方は短い文にも関わらず、どんどん大きくなっています。これは毎回これまでのやり取りを今回の質問の前にこっそり追加しているためです。この追加を行わないと、これまでの回答と無関係な回答になるために、特にプログラム作成には使えません。残念ながら質問に許される token の数には制限があり、長いやり取りになると、最初の方の部分は切り捨てられてしまいます。

質問: さらに、ふらふらと左右に動く赤い敵を 1 つ表示し、敵もオレンジ色の弾をランダムに発射するようにしてください。(1040 tokens)

回答: ゲームに敵と敵の弾を追加するために、以下のような更新を行います。このサンプルコードでは、赤い敵が左右にふらふらと動き、オレンジ色の弾をランダムに発射します。

```

html
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>シューティングゲーム</title>
  <style>
    canvas {
      background-color: #000;
      display: block;
      margin: 0 auto;
    }
  </style>
</head>
<body>
  <canvas id="gameCanvas" width="800" height="600"></canvas>
  <script>
    . . . 中略 . . .
  </script>

```

```

    </script>
</body>
</html>

```

追加された要素の説明

- ****敵オブジェクト****: 赤い四角として描画される敵は、左右に動くように設定されています。キャンバスの端に達すると移動方向が逆になります。
- ****敵の弾****: 敵からランダムに発射される弾を管理するために新しい配列 'enemyBullets' を作成しました。
- ****ランダムな弾発射****: 敵は 2% の確率でフレームごとにオレンジ色の弾を発射します。

このコードをもとに、当たり判定の実装や敵の数、動き方の追加などを行うことで、さらにゲーム性を高めることができます。(1236 tokens)

中略部分を含めて 146 行になりました。一昨年この例を作成するために、数日かけてかなり試行錯誤を行いました。やり直す度に違うやり方の回答がありました。わざわざ自機の色を青と指定しているのは、白地に白い自機を表示したことがあったからです。他のものにも色を指定しているのは、同様の理由です。シューティングゲームは宇宙での打ち合いというものが多かったので、何も指定しなくても背景が黒色になりましたが、あまり背景色を自覚していないようです。敵も味方も弾がどンドン撃てるようになってますが、大昔のインベーダーゲームのように画面内で動く弾は一つだけの回答もありました。

質問: 自分の弾が敵に当たったら白色で大きく『You win』、敵の弾が自分に当たったら赤色で大きく『You lose』と表示してゲームが終了するようにしてください。(2340 tokens)

回答: ゲームに勝敗判定を追加し、プレイヤーの弾が敵に当たったときに『You win』、敵の弾がプレイヤーに当たったときに『You lose』と表示する機能を実装します。以下はそのためのコードです。

```

html
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>シューティングゲーム</title>
  <style>
    canvas {
      background-color: #000;
      display: block;
      margin: 0 auto;
    }
  </style>
</head>
<body>
  <canvas id="gameCanvas" width="800" height="600"></canvas>
  <script>
    const canvas = document.getElementById('gameCanvas');
    const ctx = canvas.getContext('2d');

    const ship = {
      x: canvas.width / 2,
      y: canvas.height - 50,
      width: 30,
      height: 30,
      color: 'blue',
      speed: 5
    };

    const bullets = [];
    const bulletSpeed = 7;

    const enemy = {
      x: canvas.width / 2,
      y: 50,
      width: 30,
      height: 30,
      color: 'red',

```

```
        speed: 2,
        direction: 1
    };

    const enemyBullets = [];
    const enemyBulletSpeed = 5;

    let keys = {};
    let gameEnded = false;
    let message = '';

    document.addEventListener('keydown', (e) => {
        keys[e.key] = true;
    });

    document.addEventListener('keyup', (e) => {
        keys[e.key] = false;
    });

    function update() {
        if (gameEnded) return;

        // 自機の移動
        if (keys['ArrowLeft'] && ship.x > 0) {
            ship.x -= ship.speed;
        }
        if (keys['ArrowRight'] && ship.x + ship.width < canvas.width) {
            ship.x += ship.speed;
        }
        if (keys[' ']) {
            shootBullet();
        }

        // 自機の弾の更新
        for (let i = bullets.length - 1; i >= 0; i--) {
            bullets[i].y -= bulletSpeed;
            if (bullets[i].y < 0) {
                bullets.splice(i, 1);
                continue;
            }

            // 敵との衝突判定
            if (checkCollision(bullets[i], enemy)) {
                gameEnded = true;
                message = 'You win';
            }
        }

        // 敵の移動
        enemy.x += enemy.speed * enemy.direction;
        if (enemy.x <= 0 || enemy.x + enemy.width >= canvas.width) {
            enemy.direction *= -1;
        }

        // 敵の弾の発射 (ランダム)
        if (Math.random() < 0.02) {
            shootEnemyBullet();
        }

        // 敵の弾の更新
        for (let i = enemyBullets.length - 1; i >= 0; i--) {
            enemyBullets[i].y += enemyBulletSpeed;
            if (enemyBullets[i].y > canvas.height) {
                enemyBullets.splice(i, 1);
                continue;
            }

            // 自機との衝突判定
            if (checkCollision(enemyBullets[i], ship)) {
                gameEnded = true;
                message = 'You lose';
            }
        }
    }
}
```

```

function checkCollision(obj1, obj2) {
    return obj1.x < obj2.x + obj2.width &&
           obj1.x + obj1.width > obj2.x &&
           obj1.y < obj2.y + obj2.height &&
           obj1.y + obj1.height > obj2.y;
}

function shootBullet() {
    if (bullets.length === 0 ||
        bullets[bullets.length - 1].y < canvas.height - 150) {
        bullets.push({ x: ship.x + ship.width / 2 - 2.5,
                       y: ship.y,
                       width: 5, height: 10,
                       color: 'green' });
    }
}

function shootEnemyBullet() {
    enemyBullets.push({ x: enemy.x + enemy.width / 2 - 2.5,
                       y: enemy.y + enemy.height,
                       width: 5, height: 10,
                       color: 'orange' });
}

function draw() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    if (gameEnded) {
        ctx.fillStyle = message === 'You win' ? 'white' : 'red';
        ctx.font = '48px Arial';
        ctx.textAlign = 'center';
        ctx.fillText(message, canvas.width / 2, canvas.height / 2);
        return;
    }

    // 自機を描画
    ctx.fillStyle = ship.color;
    ctx.fillRect(ship.x, ship.y, ship.width, ship.height);

    // 自機の弾を描画
    for (let bullet of bullets) {
        ctx.fillStyle = bullet.color;
        ctx.fillRect(bullet.x, bullet.y, bullet.width, bullet.height);
    }

    // 敵を描画
    ctx.fillStyle = enemy.color;
    ctx.fillRect(enemy.x, enemy.y, enemy.width, enemy.height);

    // 敵の弾を描画
    for (let bullet of enemyBullets) {
        ctx.fillStyle = bullet.color;
        ctx.fillRect(bullet.x, bullet.y, bullet.width, bullet.height);
    }
}

function gameLoop() {
    update();
    draw();
    requestAnimationFrame(gameLoop);
}

gameLoop();
</script>
</body>
</html>

```

新規追加・変更点の説明

- **衝突判定**：'checkCollision'関数が追加され、弾と対象オブジェクト間の衝突を検知します。自機の弾が敵に当たった場合には「You win」、敵の弾が自機に当たった場合には「You lose」を表示し、'gameEnded'フラグを'true'にします。

- **ゲーム終了の制御**：ゲームが終了すると、メッセージを表示し、'update'関数が早期終了して動作を停止します。

この完成形は基本的な勝敗判定を行います、ゲームをリスタートできるようにする、より洗練されたメカニクスを取り入れるなど、さまざまな拡張が可能です。(1466 tokens)

最初文字の色を指定しなかったら、普通に黒色で出したので、どちらが勝ったのかわからないものができました。179行になりました。

以上の ChatGPT の最後の回答をブラウザで表示させると図 7.1 のようになりました。想像していたものと比べてどうでしょうか。ちゃんとキー操作で弾は出るし、自機を動かすこともできます。敵も動いて時々弾を降らせます。

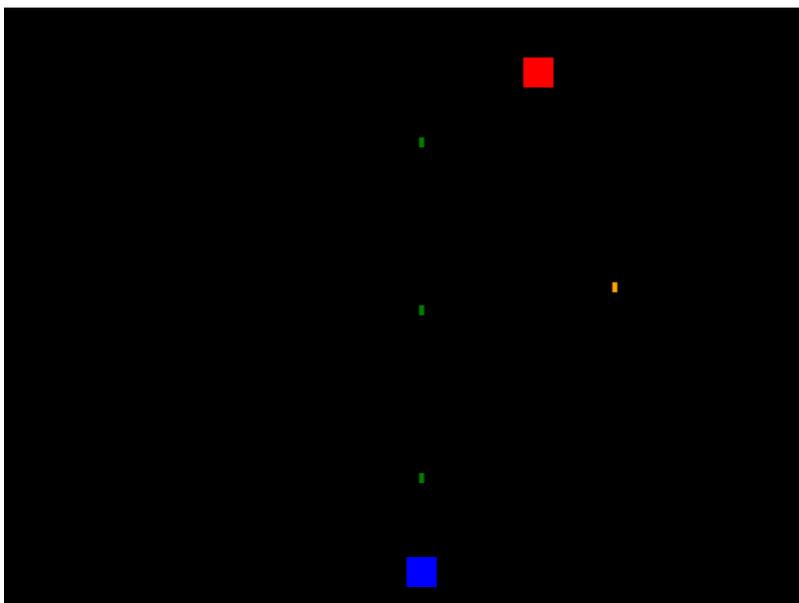


図 7.1 完成したゲームの実行

ChatGPT の利用例として、プログラム開発の補助があります。この場合の出力の大半は、文章ではなくプログラムになります。そのためにより多くのプログラム例を学習させたものが出てきました。2025 年 11 月に登場した ChatGPT-5.1-codex がその例で、次のような簡単な入力のみで、それなりのものができました。

質問: JavaScript によるシューティングゲームを作ってください。敵は一機だけ左右にのみふらふら動き弾を撃ってくるものとします。(63 tokens)

以下は、HTML と JavaScript (Canvas を使用) で作成したシンプルなシューティングゲームのサンプルコードです。

プレイヤーは画面下部にいて左右に動き、スペースキーで弾を撃てます。
敵は画面上部に一機だけいて、左右にふらふら動きます。
敵は一定間隔で弾を撃ってきます。
プレイヤーの弾が敵に当たると敵は消えます(倒した扱い)。
敵の弾がプレイヤーに当たるとゲームオーバー。

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>シンプルシューティングゲーム</title>
<style>
body {
margin: 0;
background: #000;
overflow: hidden;
```

```
}
canvas {
  display: block;
  margin: 0 auto;
  background: #111;
}
#message {
  color: white;
  text-align: center;
  font-family: sans-serif;
  margin-top: 10px;
}
</style>
</head>
<body>
<canvas id="gameCanvas" width="480" height="640"></canvas>
<div id="message"></div>

<script>
(() => {
  const canvas = document.getElementById("gameCanvas");
  const ctx = canvas.getContext("2d");

  const WIDTH = canvas.width;
  const HEIGHT = canvas.height;

  // プレイヤー設定
  const player = {
    x: WIDTH / 2,
    y: HEIGHT - 50,
    width: 40,
    height: 20,
    speed: 5,
    color: "lime",
    bullets: [],
    canShoot: true,
    shootCooldown: 300 // ms
  };

  // 敵設定
  const enemy = {
    x: WIDTH / 2,
    y: 50,
    width: 50,
    height: 30,
    color: "red",
    speed: 2,
    direction: 1, // 1は右、-1は左
    bullets: [],
    shootInterval: 1500,
    alive: true
  };

  // 弾設定
  const bulletWidth = 5;
  const bulletHeight = 10;
  const bulletSpeed = 7;

  let keys = {};

  let gameOver = false;
  let gameWin = false;

  const messageDiv = document.getElementById("message");

  function drawRect(x, y, width, height, color) {
    ctx.fillStyle = color;
    ctx.fillRect(x, y, width, height);
  }

  // プレイヤー弾の更新と描画
  function updatePlayerBullets() {
    player.bullets.forEach((b, i) => {
      b.y -= bulletSpeed;
      if (b.y + bulletHeight < 0) {
```

```
        player.bullets.splice(i, 1);
    }
  });
}

function drawPlayerBullets() {
  player.bullets.forEach(b => {
    drawRect(b.x, b.y, bulletWidth, bulletHeight, "lime");
  });
}

// 敵弾の更新と描画
function updateEnemyBullets() {
  enemy.bullets.forEach((b, i) => {
    b.y += bulletSpeed;
    if (b.y > HEIGHT) {
      enemy.bullets.splice(i, 1);
    }
  });
}

function drawEnemyBullets() {
  enemy.bullets.forEach(b => {
    drawRect(b.x, b.y, bulletWidth, bulletHeight, "red");
  });
}

// 衝突判定
function isColliding(r1, r2) {
  return !(
    r1.x + r1.width < r2.x ||
    r1.x > r2.x + r2.width ||
    r1.y + r1.height < r2.y ||
    r1.y > r2.y + r2.height
  );
}

// 敵弾発射
function enemyShoot() {
  if (!enemy.alive) return;
  enemy.bullets.push({
    x: enemy.x + enemy.width / 2 - bulletWidth / 2,
    y: enemy.y + enemy.height,
    width: bulletWidth,
    height: bulletHeight
  });
}

// プレイヤー弾発射
function playerShoot() {
  if (!player.canShoot) return;
  player.bullets.push({
    x: player.x + player.width / 2 - bulletWidth / 2,
    y: player.y - bulletHeight,
    width: bulletWidth,
    height: bulletHeight
  });
  player.canShoot = false;
  setTimeout(() => {
    player.canShoot = true;
  }, player.shootCooldown);
}

// 敵の動き更新
function updateEnemy() {
  if (!enemy.alive) return;
  enemy.x += enemy.speed * enemy.direction;
  if (enemy.x < 0) {
    enemy.x = 0;
    enemy.direction = 1;
  } else if (enemy.x + enemy.width > WIDTH) {
    enemy.x = WIDTH - enemy.width;
    enemy.direction = -1;
  }
}
```

```
    }  
  }  
  // プレイヤーの動き更新  
  function updatePlayer() {  
    if (keys["ArrowLeft"] || keys["a"]) {  
      player.x -= player.speed;  
      if (player.x < 0) player.x = 0;  
    }  
    if (keys["ArrowRight"] || keys["d"]) {  
      player.x += player.speed;  
      if (player.x + player.width > WIDTH) player.x = WIDTH - player.width;  
    }  
  }  
  // 衝突処理  
  function checkCollisions() {  
    // プレイヤー弾が敵に当たる  
    if (enemy.alive) {  
      player.bullets.forEach((b, i) => {  
        if (isColliding(b, enemy)) {  
          enemy.alive = false;  
          player.bullets.splice(i, 1);  
          gameWin = true;  
          messageDiv.textContent = "敵を倒した！ゲームクリア！";  
        }  
      });  
    }  
  }  
  // 敵弾がプレイヤーに当たる  
  enemy.bullets.forEach((b, i) => {  
    if (isColliding(b, player)) {  
      gameOver = true;  
      messageDiv.textContent = "ゲームオーバー！";  
    }  
  });  
}  
// 描画処理  
function draw() {  
  ctx.clearRect(0, 0, WIDTH, HEIGHT);  
  
  if (enemy.alive) {  
    drawRect(enemy.x, enemy.y, enemy.width, enemy.height, enemy.color);  
  }  
  
  drawRect(player.x, player.y, player.width, player.height, player.color);  
  
  drawPlayerBullets();  
  drawEnemyBullets();  
}  
// 敵撃つタイマー  
let enemyShootTimer = setInterval(() => {  
  if (!gameOver && !gameWin && enemy.alive) {  
    enemyShoot();  
  }  
}, enemy.shootInterval);  
// メインループ  
function loop() {  
  if (!gameOver && !gameWin) {  
    updatePlayer();  
    updateEnemy();  
    updatePlayerBullets();  
    updateEnemyBullets();  
    checkCollisions();  
  }  
  draw();  
  requestAnimationFrame(loop);  
}  
// キーボード操作  
window.addEventListener("keydown", e => {
```

```
keys[e.key] = true;
if (e.key === " " || e.key === "Spacebar") {
  playerShoot();
  e.preventDefault();
}
});
window.addEventListener("keyup", e => {
  keys[e.key] = false;
});
// ゲーム開始
loop();
})();
</script>
</body>
</html>
```

使い方

左右の矢印キーまたは a / d キーでプレイヤーを左右に移動します。
スペースキーで弾を撃てます。
敵は左右にふらふら動き、1.5 秒ごとに弾を撃ってきます。
当たり判定が発生すると、敵を倒すかゲームオーバーになります。

シンプルですが、基本的なシューティングゲームの動きは入っています。自由に拡張してください！
(4585 tokens)

252 行になりますが、ChatGPT-4o に同じ入力をした場合と比べると、当たり判定などが入っています。

このように JavaScript についての知識が全く無いと無理ですが、初級レベルでもあれば簡単なゲームぐらいはなんとかかなりそうです。この例のように、少しずつ望んだ形に修正していくとか、プログラムの一部についてやり方を尋ねる、と言うような使い方が無難でしょう。ただ、どのようなプログラムを作りたいのか、**完成時のイメージを持たないと使えません。**

7.2 ChatGPT をどのように捉えるか

ChatGPT は広く知られ・使われるようになってからまだ余り時間が経っていないので、一言でこれを示す言葉はありません。また ChatGPT は様々な事に使える上に、まだ新しい使い方が考えられたり、機能の追加が行われるために、何ができる・何ができないという基本的な点さえ明確ではありません。ここでは現在の時点での三木の感じていることを述べます。

7.2.1 高級かな漢字変換

ChatGPT は「高級かな漢字変換」である。昔々コンピュータがアメリカからやってきと時、これで如何に日本語を扱うかが問題になりました。日本語の文字をどのような数値に対応させるか、日本語の文字をどのように表示・印刷させるか、日本語の文字をどのように入力するか。最後の問題に関しては「かな漢字変換」で対応してきました。読みを全て入力して正しい漢字に置き換えると言う方式です。漢字には同音異字があるため、誤変換を完全に無くすことは不可能ですが、ある程度文の意味が読み取れるぐらいの長さの読みを入れると、かなり正確な変換が可能です。これが携帯端末が普及してくる頃には、短い読みを入力すると、その続きを含めて漢字に置き換えたものを示すようになりました。読みを全て入力していた時代から、一部を入力すれば良い時代になったのです。そして ChatGPT によって、短い語句を入力するだけで長い文章が出てくるようになりました。かな漢字変換が更に進化したと言えます。

ChatGPT も「かな漢字変換」と考えると、その問題点や解決法も見えてきます。ChatGPT も時には意図しない結果を出します。これは誤変換に相当します。これを避けるには、「かな漢字変換」の時に長い読みを入れたように、十分な指示を与えます。また少ない入力を長い文章にするために話を盛ってきます。大量に学習

したよくある話を入れてくるので、指示していない部分には注意します。現在のところ ChatGPT は一連のやり取りの間は、何を返したか覚えています、次回は忘れていきます。一番古い「かな漢字変換」でも、前回選択した同じ読みの漢字は忘れずに最初の候補として出てくるので、ChatGPT も見習って欲しいところです。

テレビドラマで会社の事務員の方は、ずっとパソコンに向かってお仕事をしています。ChatGPT に「いつもの文書」と指示するだけで文書ができるようになると楽になりますね。ただ「いつもの文書」と指示するだけの人に、会社が高い給料を支払ってくれる可能性は低いと思います。

7.2.2 ChatGPT は魔族である

最近人気のアニメ・漫画である「葬送のフリーレン」はファンタジーの一つですが、その主人公フリーレンが敵対するのが魔族です。魔族とは言葉を話す魔物ということになっています。つまり ChatGPT は言葉を話す魔物という捉え方です。現実の我々はこれまで言葉を話す人以外の存在を知りませんでした。既に様々な機械が言葉を発しますが、全て予め設定されたものを再生しているのに過ぎません。自由に会話ができる ChatGPT が登場すると人はどのような反応を示すかについては、実は 1966 年に作られた Eliza^{*23}が参考になります。ELIZA はコンピュータが普及していない時代に、たいへん限られた処理能力を使って、人が入力した文のパターンを読み取り、決められた応答を返すだけのシステムでした。それにも関わらず、後に ELIZA を作成したワイゼンバウムの書いた記事には、人々がいかにか素早くかつ深くそのコンピュータプログラムに感情的に没頭したかを記しています。対話の記録を見ようとするとプライバシーの侵害だとして拒んだり、対話中は部屋に一人きりにしてくれと頼んだ、などがあつたあつたそうです。

フリーレンの世界では多くの人が魔族に騙されて命を落とします。人は言葉を話す魔族に、話せば理解し合えるという気持ちを持つのですが、人ならば誰でもが持つ感情が魔族にはなく、一部の例外的な魔族を除けば、魔族は言葉を人を騙して殺すために使うのでした。人はそんなことをしたら酷いとか気の毒と感ずる事を、魔族は実際にやってみて、何も感じないのでやっぱり人の気持ちは分からん、という感じです。

結局の所、人の肉体を持たないものに、人の感情は分からないと思います。インターネットにある膨大な人の感情に関する文字情報は、人の感情の影に過ぎず、それをいくら学んでも人の感情自体を知ることはできません。それを忘れて ChatGPT に期待するのは危険でしょう。ChatGPT が魔物かどうかは分かりません。典型的な悪であれば滅ぼせば良いのですが、たいへん役に立つ存在であることは確かなのでその評価は難しい問題です。さらに会話ができるので、人は ChatGPT につくよみちゃんの例のように、存在しない感情を感じて、きっと判断を誤るでしょう。

7.2.3 黒い猫でも、白い猫でも、鼠を捕るのが良い猫だ

これは 1962 年に当時の中国のトップである鄧小平が述べた言葉で、これをきっかけに中国は共産党の考えに合わない西洋の様々なものを、積極的に取り入れて大きな発展をしました。ChatGPT が黒なのか白なのかは分かりません。もしかすると人類に大きな災いをもたらすかもしれません。それでもネズミが取れるなら良いのではないかと、という捉え方です。江戸時代の日本人の大半は歩いて移動しました。それにも関わらず、我々にはとても不可能に思えるような大旅行を庶民は行っています。例えば江戸から伊勢神宮に歩いてお参りをしていました。現在の我々は電車・バス・車などを日常的に利用し、江戸時代の人々が持っていた長時間歩く力や歩く根性を失ってしまいました。その代わりに行動範囲が広がりました。交通事故で毎年 2 千以上の人々が事故で亡くなっていますが、江戸時代に戻ろうと言う話はありません。ChatGPT を利用することにより、世界を大きく広げることができるならば、多少の犠牲は受け入れられるでしょう。

^{*23} Wikipedia の Eliza の項 <https://ja.wikipedia.org/wiki/ELIZA>

8. 深層学習

この章では近年流行の AI (Artificial Intelligence: 人工知能) でよく用いられている深層学習 (Deep Learning) について学びます。

8.1 コンピュータによる問題解決

コンピュータは、何かを入れると何かを返してくれる機械、と言う捉え方をすることができます。パソコンならばキーボードやマウスで指示を入れます。スマホならばタッチして指示します。お返しは画面への表示であったり、音声だったりします。入れたものに対して、出てきたものの間には何らかの関連があります。この関連を表すものとして代表的なものがアルゴリズムです。最大公約数を求めるアルゴリズムである「ユークリッドの互除法」が有名ですが、このアルゴリズムに従えば必ず最大公約数を求めることができます。コンピュータの動き方を決めるプログラミングは、このアルゴリズムをコンピュータで実行できる形に書き直す作業です。

どのような問題に対しても解決するアルゴリズムが存在するのだろうか？という疑問はかなり昔に理論的に否定されています。つまり理論的にアルゴリズムが存在しない問題が世の中にはあります。また良いアルゴリズムが見つからない問題、一応アルゴリズムはあるのだが、ものすごく時間がかかる問題などが多数あります。とは言うものの、コンピュータに人の代わりに働いてもらおうとすると何らかのアルゴリズムが必要です。

AI はその全てがそうではありませんが、アルゴリズム無しで問題の解決をします。アルゴリズムの代わりにデータで学習して、コンピュータが自分で問題を解く方法を発見します。人の神経組織をまねた構造を作り、人の学習のまねをするという発想はかなり昔からありました。ただこの学習がなかなかうまく行かない、学習のために膨大なデータと膨大な計算量が必要ということで、長い間お蔵入りしてました。最近になってコンピュータの計算力が超強力になり、効率よく学習させる方法が見つかってきたのでこの技術が復活して、様々な分野で使われようとしています。

AI による解決の良い点は、アルゴリズムを考えなくて良いところです。問題点は、

- 適切な学習用データが大量に必要
- 結果がきっちり出るわけではない
- なぜその結果になるのかわからない

と言うような点です。逆にアルゴリズムの良い点は、結果がきっちり出ると、なぜその結果になるのかが明確なところです。AI を利用したシステムは極端な言い方をすれば、独断と偏見で結果を出します。よって従来のアルゴリズムによって動いているシステムと同じように考えると危険です。いくら正しい結論が出ても、なぜそのような結論になるのかわからない場合、結論を出したのが人であれば疑いを持つ人が出ます。一方コンピュータが出したとか AI が出したとなると、信じる人がたくさん出てきます。よく SF に出てくるコンピュータに支配された未来世界は、もうすぐ実現するかもしれません。

8.2 機械学習

AI が学習するためのデータは、「教師あり」のもの、「教師なし」のものに分けられます。大抵は「教師あり」です。これは正解付きデータと言っても良いでしょう。教師ありデータを学習して、学習していないデータに対して正解が答えられるようになることを目指します。人の勉強とは違って、AI は学習の際に使用したデータに対しては必ず正解を答えます。学習していないデータに対してどのくらい正しい答えを出せるかが評価のポイントになります。

「教師なし」の例としては、2012 年に Google がインターネット上の様々な写真画像を処理した結果、猫を認識できるようになった、と言うものがあります。これは個々の画像に対して「猫である」とか「猫でない」

という情報を与えることなく、猫を識別できるようになりました。もちろん AI が行ったのは単なる画像の分類です。その分類された画像グループの一つを人が見たら猫ばかりだった、ということです。個々のデータに対して正解を付ける作業は大変な手間がかかることが多いので、教師なしの学習は困難ですが様々な研究が行われています。Google の作った Alpha GO*²⁴ と言うシステムが囲碁の世界チャンピオンに勝ったと言う話は有名なのでご存知だと思います。この世界チャンピオンに勝ったシステムは、過去の対戦記録を 3,000 万局分も学習していたそうです。ところが現在このシステムにボロ勝ちをするシステムがあります。囲碁などのゲームは勝ち負けが明確です。その性質を利用して、AI システム同士で対戦させて勝ち負けを自分らで判定して学習するようにして、過去の対戦記録なしに大量の学習を行った結果です。このようにシステムが正解を判定できるようにして実現した教師なし学習も有望です。

8.3 ニューラルネットワーク

ニューラルネットワークは、図 8.1 のような人などの生物の神経組織をモデル化したものです。神経細胞 (ニューロン) は、樹状突起と呼ばれる部分で他の神経細胞などからの信号を受け取り、数 mm~数十 cm の細長い軸索で他の細胞に信号を送り出します。神経細胞では化学物質を利用して信号伝達を行います。これをモデルにしたニューラルネットワークでは数値を使います。神経細胞は通常複数の細胞から信号を受け取りますが、どの細胞からの信号も同じ扱いにしている訳ではありません。同様にニューラルネットワークでも、受け取った数値に適当な数を掛け算してから合計します。その結果を活性化関数というもので計算した結果を次へ送ります。ニューラルネットワークでの学習の実体は、この掛け算する数 (係数) を求めることです。

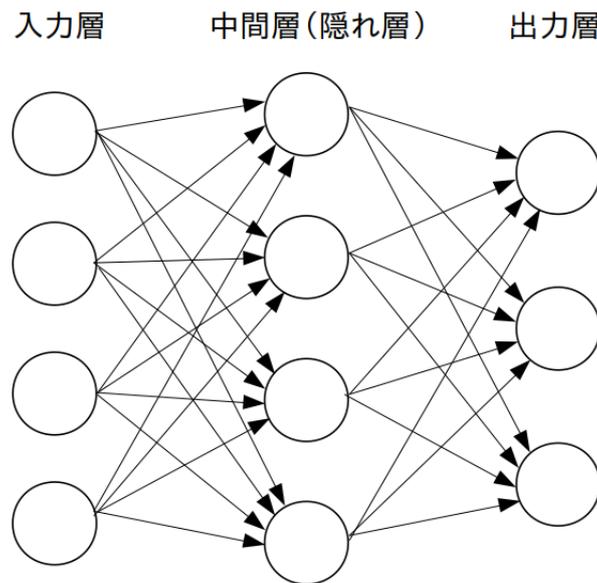


図 8.1 ニューラルネットワーク

図 8.1 では、入力層に 4 つのニューロンがありますが、この数は入力の数に対応しています。数が多い例としては、画像の入力があります。我々が目で物を見るとき、目の中の視神経が見えているものを細かく分割して脳に送っています。同様にニューラルネットワークで物体の認識をさせる場合は、画素ごとにニューロンを割り当てます。800 万画素のカメラにそのまま対応させようとする入力層に 800 万個のニューロンが必要となりますが、さすがに多すぎるので 100×100 程度に間引いて扱うのが普通です。処理の都合から考えると少

*²⁴ 私が恐ろしいと思ったのは、Google はこの囲碁プログラムの開発に 900 億円かけたという話を聞いたときです。それまでに囲碁対戦プログラムの研究や開発にかけられた費用を遥かに超える額を、世界チャンピオンに勝てるめどもないに出すと言う Google に、勝てる日本の会社はあるのだろうか。

ないほどよいのですが、間引きしすぎると何の画像か人が見てもわからなくなります。人が見てもわからないものは、ニューラルネットワークで処理してもたいてい識別できません。

図 8.1 では、中間層は 1 段で 4 つのニューロンから構成されていますが、ここを何段にするか、ニューロンを何個にするかは決まっています。段数や個数を増やすと処理に時間がかかり、学習がうまく行かない可能性が高くなりますが、段数や個数が足りないといくら学習させても正答率が上がりません。似たような事例を参考に段数と個数を決めてやってみて、だめならば少し変更してやってみるを繰り返すしかありません。

図 8.1 では、出力層に 3 個のニューロンがありますが、これは結果の種類の数で決まります。例えば学生さんの顔写真を入れて、可愛さのみを求めると、出力層はニューロンが 1 個になります。同時に美しさ、賢そうか、優しそうかも求めるならば 4 個になります。

図 8.1 のニューラルネットワークには矢印が 28 本 ($= 4 \times 4 + 4 \times 3$) あります。この矢印の数だけ係数があり、学習によってこれを求めなければなりません。残念ながら二次方程式のような解法はありません。とりあえず係数に適切な数値を設定してやってみて、少しずつ正しい答えに近づくように修正していくと言うような解法になります。山登りに例えると、足元の地面を見てより高い方へ向かって進んで行けば、そのうち山頂にたどり着けるだろうと言う感じです。繰り返してもあまり改善が見られなくなったところで学習終了とします。残念な結果になったら、データを増やすとか中間層の構成を変更することになります。データや中間層の構成に問題がなくても失敗することがあります。山で遭難しそうになったら、麓(ふもと)を目指して低い方へと進むのですが、盆地にたどり着くとどちらを向いても上り坂になってしまい、進む方向がわからなくなります。同様なことがニューラルネットワークの学習にも生じることがあります。現実の山と違って地図もないので盆地にハマっているかどうかすら通常はわかりません。

この問題に対して例えば**確率勾配降下法**と言う手法を用いて盆地にハマらないようにしています。学習用データをランダムに小分けして学習させる方法です。全員で行き先を決めると、行き先が平均化されて盆地にハマるかもしれないが、多数の小グループであればその中にはとんでもない方向へ行くグループもあるだろうし、もしかすると麓への道を見つけ出すグループが出てくるかもしれないと言う感じです。

画像認識などでより良い認識率を実現するために**畳み込みニューラルネットワーク (CNN)** と言うものも考えられました。確か猫で行われた実験だったと思いますが、生まれたときから縦縞ばかりの環境で育てたところ縦縞にしか反応しなくなり、頭の中の神経を調べたところ縦縞のみに反応する部分が見つかったが、横縞に反応する部分はなかったというものです。我々は様々な形のものを見て育ちますから、頭の中には縦縞に反応する神経だけでなく、横縞や角などの各種部分に対応する神経があるでしょう。そういうものの組み合わせとして物を認識している可能性があります。畳み込みニューラルネットワークでは画像から複数のフィルターを使って複数の特徴的な部分を探しその結果を利用して学習を行います。またどのような特徴が有効なのかは分かりませんので、並行してフィルターの内容も学習します。

英語を日本語に翻訳させる、株価の予想をさせたい、と言うような場合も図 8.1 のようなニューラルネットワークでは対応が難しくなります。明日の株価を予想させるために、前日の株価だけではなかなか当たりません。ある程度の幅 (例えば一週間) を持った過去の株価を元に予想した方が良いでしょう。そして、単純に過去 7 日分の株価データではだめで、株価の推移も考慮してもらわないといけません。英語や日本語もどの単語があったでは意味が取れません。同じ単語でも先頭に出てくれば大抵主語ですので語順は重要です。**再帰的ニューラルネットワーク (RNN)** では、7 日分の株価から明日の株価を予想する場合、図 8.1 のようなニューラルネットワークを 7 つ用意する代わりに、一つのニューラルネットワークに日にちと株価のセットのデータを与えて学習させます。このやり方ならば 100 日前の株価からだって入れられますが、あまり欲張ると学習できなくなります。

Deep Q Network (DQN) は強化学習用のニューラルネットワークです。強化学習は教師なし学習の一種と言われていますが、例えば迷路から抜け出す道筋を学習するような場合に使われます。現在位置の様子を見て、前進・後退・右へ・左へと選択をします。そのうち出口にたどり着いたらこれが正解として学習します。一番遠回りの道を選ぶようになっては困りますから、より短いコースを選択するように学習させます。囲碁の

学習も同様に強化学習で行いました。現在これを車の自動運転に利用することが検討されています。車が自分の周囲を見て、ちゃんと道路の上を走るようにするためです。

8.4 様々なフレームワーク

深層学習のために様々なフレームワーク (枠組み) が作られています。フレームワークを利用することにより容易に深層学習を行うシステムを作ることができます。代表的なものとして、Google が作った Tensor Flow、Amazon が作った MxNet、Microsoft が作った CNTK、Preferred Networks が作った Chainer などがあります。どのフレームワークを使うにしろ Python というプログラミング言語に関する知識が必要になります。Python 自体は深層学習のためのプログラミング言語と言うものではなく、普通のプログラムが書けるものです。将来 AI システムを作るお仕事をするならば Python を勉強しておくべきですが、他のプログラミング言語とは少し書き方が違うので、将来未定の人に勧めるのはどうかなあ、と現在は思っています。

有名な会社がそれぞれ別のフレームワークを打ち出しているのは、AI の中核とも言える深層学習での覇権を狙っているからです。多くの利用者があれば、多くの使用例が生まれ、大学などの授業で使用例が紹介され、そういう授業を受けた学生が会社で同じようにやってみようとする訳です。このテキストでは Chainer を使用することにしました。他のフレームワークが超有名会社が作っているのに対して、Preferred Networks なんて聞いたことがない会社でしょうが、実はこの会社は日本の会社です。よってトヨタ自動車など日本のメーカーや大学などとの共同開発の例などがあるようです。こうして令和元年度は国産のフレームワークに賭けてみたのですが、その後 Chainer の開発が終了しました。来年度は別のフレームワークにしようと思います。(毎年同じことを書いている。)

8.5 画像認識プログラムの例

株式会社明治が 40 年以上前から販売している「きのこの山」というお菓子があります。姉妹品として「たけのこの里」と言うお菓子もあります。この両者を見分けるのは幼児でも簡単な問題ですが、コンピュータではなかなか面倒なプログラムを書かなければなりません。この 2 種類のお菓子を深層学習で学習させたモデルを利用して見分けるプログラム^{*25}の例を元に、小物の文房具 4 種類を見分けるものに直したものがリスト 1 です。

リスト 1 見分けるプログラムの例

```
1 # -*- coding: utf-8 -*-
2 # CQ 出版「インターフェース」2017 年 8 月号 p.41-43 のものを一部改変
3 import argparse
4 import os
5 import numpy as np
6 from PIL import Image
7 from PIL import ImageEnhance
8
9 import chainer
10 import chainer.functions as F
11 import chainer.links as L
12 import chainer.initializers as I
13 from chainer import training
14 from chainer.training import extensions
15
```

^{*25} オリジナルは、CQ 出版社「インターフェース」2018 年 8 月号 p.41-43 の牧野浩二、西崎博光による「ラズパイでディープ・ラーニング初体験 (2)・・・画像認識」による。入力部分など一部改変している。

```
16 # 畳み込みニューラルネットワークの定義
17 class CNN(chainer.Chain):
18     def __init__(self, n_units, n_out):
19         w = I.Normal(scale=0.05) # モデルパラメータの初期化
20         super(CNN, self).__init__(
21             conv1=L.Convolution2D( 3, 16, 5, 1), # 1層目の畳み込み層
22             conv2=L.Convolution2D(16, 32, 5, 1), # 2層目の畳み込み層
23             conv3=L.Convolution2D(32, 64, 5, 1), # 3層目の畳み込み層
24             l4=L.Linear(None, n_out, initialW=w), #クラス分類用
25         )
26
27     def __call__(self, x):
28         # 最大値プーリングは 2× 2,活性化関数はReLU
29         h1 = F.max_pooling_2d(F.relu(self.conv1(x)), ksize=2, stride=2)
30         h2 = F.max_pooling_2d(F.relu(self.conv2(h1)), ksize=2, stride=2)
31         h3 = F.max_pooling_2d(F.relu(self.conv3(h2)), ksize=2, stride=2)
32         # 9x9,64ch
33         return self.l4(h3)
34
35 # 画像の大きさを指定サイズに切り詰める
36 def crop(img, size):
37     w, h = img.size
38     assert w >= size or h >= size, "画像サイズが小さすぎます"
39     if w > h :
40         p = (w - h) / 2
41         box = (p,0,p+h,h)
42     else :
43         p = (h - w) / 2
44         box = (0,p,w,p+w)
45     return img.crop(box).resize((size, size))
46
47 # 画像の 4つの角の明るさを元に、全体の明るさを調整する
48 def doBright(img):
49     imgrgb = img.convert('RGB')
50     r,g,b = imgrgb.getpixel((0,0))
51     s=r+g+b
52     r,g,b = imgrgb.getpixel((99,99))
53     s+=r+g+b
54     r,g,b = imgrgb.getpixel((0,99))
55     s+=r+g+b
56     r,g,b = imgrgb.getpixel((99,0))
57     s+=r+g+b
58     er=1+(2400-s)/1600
59     return ImageEnhance.Brightness(img).enhance(er)
60
61 def main():
62     # オプション処理
63     parser = argparse.ArgumentParser(description='Chainer example: MNIST')
64     parser.add_argument('--model', '-m', default='model',
65                         help='Resume the training from snapshot')
```

```

66 parser.add_argument('--unit', '-u', type=int, default=1000,
67                       help='Number of units')
68 args = parser.parse_args()
69
70 model = L.Classifier(CNN(args.unit, 4))
71 chainer.serializers.load_npz(args.model, model)
72 hantei = ['だるま型', '画鋏', 'クリップ', '割りピン']
73 np.set_printoptions(precision=6, floatmode='fixed', suppress=True)
74 for imgname in [f for f in sorted(os.listdir('testdata')) if ('jpg' in f)]:
75     img = Image.open(os.path.join('testdata', imgname))
76     img = np.asarray(doBright(crop(img, 100)))
77     img = img.transpose(2, 0, 1)
78     x = chainer.Variable(np.array([img]).astype(np.float32))
79     y = model.predictor(x)
80     c = F.softmax(y).data.argmax()
81     s = F.softmax(y)
82     print('{:<15} は{}です。{}'.format(imgname, hantei[c], s.data))
83
84 # main()関数を実行する
85 main()

```

このプログラムは Python というプログラミング言語で記述されています。さらに細かいことを言えば ver.3 の Python です。ver.2 の Python ので書かれたプログラムもまだよく使われているのですが、微妙に書き方が異なります。

3-14 行 `import` という語が使われていますが、これは既にあるライブラリの読み込みを指示しています。ライブラリは Excel の関数のようなものの集まりで、Python には様々な用途のライブラリがあります。ここでは数値処理、画像処理、そして `chainer` などのライブラリを読み込んでいます。

17-33 行 ここで `chainer` で使用するニューラルネットの定義をしています。17 行目から `CNN` というクラスの指定が始まり、その中身は 18-25 行目と 27-33 行目の部分に分かれています。意外に短いですがこの書き方を理解できれば、自分の好きなニューラルネットを作ることができるようになります。Python ってすごいなあと思われるかもしれませんが、この程度の行数で書けるのは、最初の部分で `chainer` のライブラリを読み込んでいるためで、Python が偉いわけではありません。プログラムの行の先頭が段々と右へ後退し、また左へもどる様子が見られると思いますが、これは Python の特徴的なところです。通常のプログラミング言語では `()` や `{}` で囲うところを、Python ではその部分を右へ後退させて示します。

36-45 行 ここで `crop` という関数を定義しています。画像処理ライブラリの助けを借りて、この関数は一つの画像データを $size \times size$ に修正します。 $size$ の具体的な値は 76 行目で `crop` を呼び出しているところで 100 だとわかります。

48-59 行 またここで `doBright` という関数を定義しています。画像処理ライブラリの助けを借りて、縮小した画像の四隅の明るさを調べて、その平均をもとに全体の明るさの修正を行っています。

61-82 行 ここでプログラムの本体となる `main` 関数を定義しています。関数は定義だけでは動かないので、85 行目で呼び出しています。

63-68 行 これはプログラム起動時に指定した設定の取り込みを行っています。学習したモデルのファイル名と、`unit` の数を取り込んでいます。

70-71 行 ここで既に定義した `CNN` クラスを実体化し、指定したモデルをそこに読み込んでいます。

72 行 モデルの出力するカテゴリー (分類) の数が 4 つなので、それに対応するメッセージを 4 つここで用意

しています。

74-78 行 ここで「testdata」ディレクトリ内の jpg のファイルを順番に取り出し、大きさを揃え、ニューラルネットの入力用の形式に変換しています。取り出された画像が最終的に変数 x に入ります。

79 行 ここで変数 x の内容についてニューラルネットの処理を行い結果を変数 y に入れています。

80-81 行 得られた結果から、一番大きな値となったカテゴリーの番号を変数 c に、全てのカテゴリーの値を変数 s に入れています。

82 行 変数 c の値を元にメッセージを表示しています。

結局文房具の画像を識別するプログラムとして特有の部分は、72 行目しかありません。ここさえ直して、「かわいい女子大生」、「きれいな女子大生」、「賢そうな女子大生」、「優しそうな女子大生」の画像を大量に用意して学習したモデルを使えば、女子大生識別プログラムの出来上がりとなります。

8.6 画像認識プログラムの実行

ここでは前節で説明したプログラムを実際に動かしてみます。コマンドをキーボードから入力して実行するという一番原始的な方法を説明します。

1. 「教材フォルダ」から「deep1」のフォルダを丸ごとデスクトップにコピーします。単にドラッグすればコピーされます。
2. コピーした「deep1」フォルダをダブルクリックして開きます。開いたウィンドウの「ツール」メニューの中の「現在のフォルダで端末を開く」を選択すると「QTerminal」が起動します。(メニューを使う代わりに **F4** キーを押しても同様になります。)「miki@mars:~/Desktop/deep1\$」のようなプロンプトが表示されますので、以下のようなコマンドを入力します。コマンドの最後には **Enter** を押します。
3. 「source .venv/bin/activate」と入力します。Python は現在もどんどん改良が進み、毎年のように大きな改訂がなされています。おかげで最新の Python では Chainer は動かなくなっています。同様の問題は他の Python で作られたアプリでも生じており、その対策として昔の環境を仮想的に作りその中でアプリを動かすという事が行われています。これはその仮想的な環境を起動するコマンドです。
4. 「python test.py -m dgkw.model」と入力します。Python で test.py を実行します。test.py は前節で説明したプログラムが入っているファイルの名前です。-m dgkw.model で学習済みモデルのファイル名 (dgkw.model) を指定しています。すると、リスト 2 のようになります。

リスト 2 画像認識の結果

```

1 miki@mars:~/Desktop/deep1$ python test.py -m dgkw.model
2 d001.jpg は「だるま型」です。[[0.999976 0.000000 0.000024 0.000000]]
3 d002.jpg は「だるま型」です。[[0.999935 0.000000 0.000065 0.000000]]
4 d003.jpg は「だるま型」です。[[0.999971 0.000000 0.000029 0.000000]]
5 d01.jpg は「クリップ」です。[[0.000000 0.000000 1.000000 0.000000]]
6 g001.jpg は「画鋏」です。[[0.000000 1.000000 0.000000 0.000000]]
7 g002.jpg は「クリップ」です。[[0.014563 0.070214 0.915223 0.000000]]
8 g003.jpg は「画鋏」です。[[0.000000 0.999995 0.000005 0.000000]]
9 k001.jpg は「クリップ」です。[[0.000000 0.000000 1.000000 0.000000]]
10 k002.jpg は「クリップ」です。[[0.000000 0.000000 1.000000 0.000000]]
11 k003.jpg は「クリップ」です。[[0.030744 0.000760 0.968496 0.000000]]
12 w001.jpg は「クリップ」です。[[0.000000 0.000000 1.000000 0.000000]]
13 w002.jpg は「画鋏」です。[[0.000000 1.000000 0.000001 0.000000]]
14 w003.jpg は「画鋏」です。[[0.000000 0.999995 0.000005 0.000000]]

```

2-13 行がプログラムの出力で、「testdata」ディレクトリに入っていたテスト画像を認識した結果が

並んでいます。行の先頭が画像ファイル名で、表4の通りに付けています。ファイル名の後に認識結果があり、その後にある [] に囲まれた4つの数字は、与えられた画像をどのくらいの精度で認識したかを示しています。残念ながら11-13行目は誤認識していますが、数字を見ても思いっきり間違えていることがわかります。

表4 画像認識の認識対象とファイル名

| 名称 | だるま型画鋲 | 画鋲 | クリップ | 割りピン |
|-------|---|---|--|---|
| 画像例 |  |  |  |  |
| ファイル名 | d~.jpg | g~.jpg | k~.jpg | w~.jpg |

このようにして、事前に用意した学習済みモデルで事前に用意した画像の認識はできましたが、自分で用意した画像を認識させる際は、次のような手順となります。

1. スマホやタブレットで認識させたい文房具を撮影します。一つ一つ撮影するのも手間ですので、混ぜて複数個まとめて撮影しても構いません。ただ後で画像を切り分ける必要があるので重ならないように撮影します。また余分なものが入らないように、白い紙の上で、影があまり映らないようにします。この辺りの条件は学習させた画像と同じ条件にすれば良いので、白っぽいものを学習させるために背景は全て黒色にした場合は、背景は黒色にしないと認識率が下がります。
2. 撮影した写真画像をパソコンに送ります。USB ケーブルで接続して送る、メールに添付して送るなどの方法があります。
3. 写真画像を Windows に標準装備されている「ペイント」で開きます。
4. 文房具を一つだけ  (四角形選択) で囲みます。できるだけ正方形になるように「ペイント」のウインドウの一番下のところに囲いの大きさがほぼ同じ (10px ぐらいは違っていても良い) になるようにします。縦と横の大きさは 100px 以上にしないと認識プログラムにはねられますのでご注意ください。そして **[Ctrl]+X** で切り取ります。(コピーでは後でどれを取り出したのかわからなくなる。) もう一つ「ペイント」を起動し、そちらを選択してから **[Ctrl]+V** で貼り付けます。**[トリミング]** をクリックすると貼り付けた画像以外の部分は削除されます。そしてこれを「ファイル」メニューの「名前を付けて保存」で保存します。形式は「JPEG 画像」にします。名前はどの文房具なのか簡単に見分けられるようなものにします。
5. mars のデスクトップにある「deep1」を開き、その中の「testdata」を開いたところへ、作成した画像ファイルをコピーします。
6. QTerminal が先程の実行した後のままであれば、**[↑]** で python で始まる行を呼び出し **[Enter]** を押せば再度認識プログラムを動かすことができます。

8.7 学習プログラムの例

ここでは4種類の文房具を学習させる際に使用したプログラム(リスト3)の説明をします。

リスト3 画像学習プログラムの例

```
1 # -*- coding: utf-8 -*-
2 import argparse
```

```
3 import os
4 import sys
5
6 import chainer
7 import chainer.functions as F
8 import chainer.links as L
9 import chainer.initializers as I
10 from chainer import training
11 from chainer.training import extensions
12
13 # 畳み込みニューラルネットワークの定義
14 class CNN(chainer.Chain):
15     def __init__(self, n_units, n_out):
16         w = I.Normal(scale=0.05) # モデルパラメータの初期化
17         super(CNN, self).__init__(
18             conv1=L.Convolution2D(3, 16, 5, 1, 0), # 1層目の畳み込み層(フィルタ数は16)
19             conv2=L.Convolution2D(16, 32, 5, 1, 0), # 2層目の畳み込み層(フィルタ数は32)
20             conv3=L.Convolution2D(32, 64, 5, 1, 0), # 3層目の畳み込み層(フィルタ数は64)
21             l4=L.Linear(None, n_out, initialW=w), # クラス分類用
22         )
23
24     def __call__(self, x):
25         # 最大値プーリングは2x2, 活性化関数はReLU
26         h1 = F.max_pooling_2d(F.relu(self.conv1(x)), ksize=2, stride=2)
27         h2 = F.max_pooling_2d(F.relu(self.conv2(h1)), ksize=2, stride=2)
28         h3 = F.max_pooling_2d(F.relu(self.conv3(h2)), ksize=2, stride=2)
29         # 9x9, 64ch
30         return self.l4(h3)
31
32 def main():
33     # オプション処理
34     parser = argparse.ArgumentParser(description='Chainer example: MNIST')
35     parser.add_argument('--batchsize', '-b', type=int, default=100,
36                         help='Number of images in each mini-batch')
37     parser.add_argument('--epoch', '-e', type=int, default=20,
38                         help='Number of sweeps over the dataset to train')
39     parser.add_argument('--gpu', '-g', action='store_true',
40                         help='GPU ID (negative value indicates CPU)')
41     parser.add_argument('--model', '-m', default='test.model',
42                         help='Name of model file')
43     parser.add_argument('--unit', '-u', type=int, default=1000,
44                         help='Number of units')
45     args = parser.parse_args()
46
47     print('# unit: {}'.format(args.unit))
48     print('# Minibatch-size: {}'.format(args.batchsize))
49     print('# epoch: {}'.format(args.epoch))
50     print('# GPU: {}'.format(args.gpu))
51     print('')
52
```

```
53 train = []
54 label = 0
55 print('loading dataset')
56 for c in sorted(os.listdir('train')):
57     print('class: {}, class id: {}'.format(c, label))
58     d = os.path.join('train', c)
59     imgs = os.listdir(d)
60     for i in [f for f in imgs if ('jpg' in f)]:
61         train.append([os.path.join(d, i), label])
62     label += 1
63 print('')
64
65 train = chainer.datasets.LabeledImageDataset(train, '.')
66 train, test = chainer.datasets.split_dataset_random(train, args.unit)
67
68 model = L.Classifier(CNN(args.unit, 4))
69 if args.gpu:
70     gpu = 0
71     chainer.cuda.get_device(gpu).use()
72     model.to_gpu()
73 else :
74     gpu=-1
75
76 # optimizer の設定
77 optimizer = chainer.optimizers.Adam()
78 optimizer.setup(model)
79
80 # 学習データ用のイタレータ
81 train_iter = chainer.iterators.SerialIterator(train, args.batchsize)
82 # 評価データ用のイタレータ
83 test_iter = chainer.iterators.SerialIterator(test, args.batchsize,
84                                             repeat=False, shuffle=False)
85
86 updater = training.StandardUpdater(train_iter, optimizer, device=gpu)
87 trainer = training.Trainer(updater, (args.epoch, 'epoch'), out='logs')
88 trainer.extend(extensions.Evaluator(test_iter, model, device=gpu))
89
90 # 計算グラフ
91 trainer.extend(extensions.dump_graph('main/loss'))
92 trainer.extend(extensions.LogReport())
93 # エポック経過ごとの損失の値を表示したりグラフ化する
94 trainer.extend(
95     extensions.PlotReport(['main/loss', 'validation/main/loss'], 'epoch',
96                          file_name='loss.png'))
97 trainer.extend(
98     extensions.PlotReport(['main/accuracy', 'validation/main/accuracy'],
99                          'epoch', file_name='accuracy.png'))
100 trainer.extend(extensions.PrintReport(
101     ['epoch', 'main/loss', 'validation/main/loss',
102     'main/accuracy', 'validation/main/accuracy', 'elapsed_time']))
```

```

103
104     # 学習開始
105     trainer.run()
106
107     # モデルをCPU 対応へ
108     if args.gpu:
109         model.to_cpu()
110     # 保存
111     print('save the trained model: {}'.format(args.model))
112     chainer.serializers.save_npz(args.model, model)
113
114 main()

```

画像認識プログラムに比べるとこちらの方が 1.5 倍ぐらい長くなります。学習の過程をグラフ化する部分の他、画像認識プログラムと共通する部分もかなりあります。

2-30 行 この部分は同じです。つまり同じライブラリーを取り込んで、同じモデルを定義しています。

32-113 行 プログラムの本体となる main を定義しています。画像は適切な大きさに変換済みとしているので、画像認識プログラムにあった crop 関数の定義部分がありません。

34-51 行 ここでプログラム起動時に指定した設定を取り込んで表示しています。学習した結果を保存するファイル名その他、学習のやり方にかかわる設定を取り込みます。

53-63 行 「train」ディレクトリーにある学習用データを変数 train に取り込みます。学習する画像は同じカテゴリのものは同じディレクトリーに入っていると想定しています。

68-74 行 ここで既に定義した CNN クラスを実体化し、GPU (Graphical Processing Unit) が指定されている場合は、さらに GPU 用の形式に変換しています。

77-88 行 ここでさらに学習用の設定を行っています。

91-102 行 学習がどのくらい行われたかをグラフ化したり、学習中に途中経過を表示するための設定をしています。

105 行 設定が全て終わったのでここで学習をします。

108-109 行 GPU を利用して学習した場合は、ここで GPU を使用しない形式のモデルに戻します。

111-112 行 学習済みのモデルを保存します。

こちらのプログラムも学習する画像によって変わるところはありません。この例では画像を 4 つに分類していますが、より多くの分類を行う場合も変更箇所はわずかです。

8.8 学習プログラムの実行

ここでは前節のプログラムを実際に動かしてみます。やり方は画像認識プログラムの実行とほぼ同じです。

1. 「教材フォルダ」から「deep2」のフォルダを丸ごと mars のデスクトップにコピーします。
2. コピーした「deep2」フォルダをダブルクリックして開き、「QTerminal」を起動します。
3. 「source .venv/bin/activate」と入力して仮想環境を起動します。
4. 「python learn.py -u 12 -e 200 -m dgkw.model」と入力して学習プログラムを実行します。するとリスト 4 のようになります。

リスト 4 画像の学習

```

1 miki@mars:~/Desktop/deep2$ python learn.py -u 12 -e 200 -m dgkw.model
2 # unit: 12
3 # Minibatch-size: 100

```

```

4 # epoch: 200
5 # GPU: False
6
7 loading dataset
8 class: d, class id: 0
9 class: g, class id: 1
10 class: k, class id: 2
11 class: w, class id: 3
12
13 epoch          main/loss    validation/main/loss  main/accuracy  validation/main/
          accuracy  elapsed_time
14 8              276.14      348.032              0.25           0.25
          1.35486
15 16             314.198      327.407              0.25           0.208333
          2.6923
16 25             303.905      251.404              0.32           0.333333
          4.10556
17 33             236.177      162                  0.42           0.291667
          7.07345
18 41             140.505      89.0948              0.34           0.375
          9.25739
19
20      中略
21
22 J158           2.22786      10.5679              0.84           0.333333
          41.0549
23 166            2.59906      9.6416               0.82           0.291667
          44.1917
24 175            1.38882      10.2373              0.84           0.208333
          46.9378
25 183            1.50393      9.23868              0.75           0.333333
          49.4368
26 191            0.628879     8.47523              0.76           0.416667
          51.4903
27 200            0.0181208    8.52638              1              0.5
          54.3045
28 save the trained model: dgkw.model

```

約 54 秒でこれらの結果が得られます。なお QTerminal の画面の幅を広げないとこのように途中で改行が入りますので、ちょっと見にくい表示になります。「loss」(損失、エラーの量)の値が小さく、「accuracy」(正確さ)の値が 1 に近いほど良いのですが、この例のように途中でまた悪い方へ変わることもあります。名称に validation がついた方は、学習用データを別の形で取り出した評価用データで測定した結果です。この例の場合、最終的には学習データに対しては 100% の正確さになっていますが、評価用データに対しては 54.3% の正確さですからほぼ半分は外れです。この学習の様子をグラフ化したものが「logs」ディレクトリーの中にあります。例えば「accuracy.png」をダブルクリックするとグラフが表示され、学習用データでの正確さは上がっていますが、評価用の方はあまり上がっていない様子が見られると思います。

- 「python test.py -m dgkw.model」で学習したモデルで画像の認識を行うことができます。結果はリスト 5 のようになります。

リスト 5 画像学習の結果の確認

```

1 miki@mars:~/Desktop/deep2$ python test.py -m dgkw.model
2 d001.jpg          は「だるま型」です。 [[0.528753 0.000000 0.471225 0.000022]]
3 d002.jpg          は「だるま型」です。 [[0.998545 0.000000 0.001450 0.000005]]
4 d003.jpg          は「クリップ」です。 [[0.482547 0.000000 0.517442 0.000011]]
5 g001.jpg          は「だるま型」です。 [[0.991425 0.008575 0.000000 0.000000]]
6 g002.jpg          は「割りピン」です。 [[0.002321 0.000000 0.000001 0.997678]]
7 g003.jpg          は「割りピン」です。 [[0.079477 0.000004 0.000028 0.920492]]
8 k001.jpg          は「クリップ」です。 [[0.000000 0.000000 1.000000 0.000000]]
9 k002.jpg          は「クリップ」です。 [[0.000000 0.000002 0.999899 0.000099]]
10 k003.jpg         は「クリップ」です。 [[0.000000 0.000000 0.999995 0.000005]]
11 w001.jpg         は「だるま型」です。 [[1.000000 0.000000 0.000000 0.000000]]
12 w002.jpg         は「クリップ」です。 [[0.002449 0.000000 0.996968 0.000583]]
13 w003.jpg         は「だるま型」です。 [[0.999981 0.000009 0.000010 0.000000]]

```

残念な結果になっていますが、学習データが少ないのか学習が足りないのでしょうか。ランダムで与えた初期値の影響も大きく、同じデータかつ同じパラメタで学習しても、認識結果がかなり違うこともあります。

矢印キーで前に入力したコマンドを呼び出して一部修正することもできます。学習の際の「-u 12 -e 200」の数字を変更して、再度学習をさせることができます。u の値はある程度大きい方が良いでしょうが、学習データの数より大きな値にすることができません。e の値を大きくすると学習の回数を増やすことができます。回数が足りないとだめですが、増やせば良くなるとは限りません。過学習と言う状態になると、学習用データに適応しすぎて、それ以外のデータに対しての応答がだめになってしまいます。また回数を増やせばそれに比例して時間もかかるようになります。

学習にかかる時間を短縮するために GPU などが用いられます。GPU は本来動画の処理 (特に動きの激しいリアルなゲームなど) のために作られたものですが、比較的安価に入手できるために最近のスーパーコンピュータでも使われています。さらに学習用であれば、それほど高精度の計算は必要ないことも判ってきたので、専用の処理装置も使われるようになっていきます。mars には Nvidia 社製の GeForce RTX 4060 Ti という GPU が付いています。その内部には 4,352 個の演算器があり、全体で 22TFLOPS^{*26}の計算ができます。これを学習の際に使う場合は、「-g」を追加します。先程は約 54 秒かかりましたが、3 秒程度で終わるようになります。

自分で用意した学習データを用いて学習させる場合には、「moto」ディレクトリの中にある「d」ディレクトリにだるま型の画像、「g」ディレクトリに画鋏の画像、「k」ディレクトリにクリップの画像、「w」ディレクトリに割りピンの画像を入れます。既に入っている画像はそのままにしても良いし、置き換えても良いです。そして「python crop.py -i moto -o train」を実行すると、「moto」ディレクトリに入っていた画像ファイルが、100×100 の大きさに整形され、明るさの調整もされて「train」ディレクトリに入ります。既に「train」ディレクトリに入っていたファイルは削除されますのでご注意ください。そして learn.py を実行すれば学習が行われます。

8.9 学習用データを増やす

とにかく学習用データを増やさないと精度良く認識することができません。一方学習用データを大量に用意するのは費用や手間がかかることは確かで、いかに効率よく用意するかが重要になります。効率の良い収集方

^{*26} FLOPS は 1 秒間に何回小数点付きの数の計算ができるかというもの。22TFLOPS は 1 秒間に 22 兆回計算できることとなります。

法はデータによって異なります。消費者の行動データのようなものは、スーパーやコンビニのレジの情報からも得られます。これらは既に商品発注や在庫管理システムで利用されていますので、関係者であれば容易に入手することができるでしょう。一部のコンビニで行われていた、レジで入力したお客さんの年齢のデータもあれば、より興味深い結果が得られるかもしれません。ただ個人情報保護には注意しなければなりません。かわいい女子大生の顔写真を集めるために、現マネ棟の入り口に密かにカメラを設置して撮影などすると問題になるでしょう。ただ密かに撮影して、分類して、学習させて、学習結果だけ残した場合、どのような画像を用いたかは分かりません。つまり、いつの間にか自分の個人情報を学習データとして使われても、本人は気が付かないので注意が必要です。と言われても、対策はAIの得意な企業にはできるだけ近寄らない、でしょうか？

画像認識の場合、認識する対象は様々な向きや大きさであることが普通です。深層学習が使われる以前の画像認識では、認識処理に入る前の前処理として、撮影した画像に映るものの向きや大きさなどを揃えていました。深層学習を使う際にも前処理をした方が効果的ですが、学習データに逆の処理を行うことがよくあります。つまり撮影した学習用データを回転させたり、縮小したのも学習させることにより、回転や縮小に強くなりますし、このようなデータの水増しは機械的にできるので手間もかかりません。学習用データの機械的な収集と効率的な学習データの水増しは、深層学習の精度を上げるための大事なポイントです。

8.10 水増し学習プログラムの実行

ここでは実際に水増したデータをもとに学習プログラムを実行し、その効果を確認します。

1. 「教材フォルダ」から「deep3」のフォルダを丸ごと mars のデスクトップにコピーします。
2. コピーした「deep3」フォルダをダブルクリックして開きます。そして「QTerminal」を起動します。
3. 「source .venv/bin/activate」と入力して仮想環境を起動します。
4. 「python cropmod.py -i moto -o train」と入力すると moto ディレクトリーの中の画像ファイルを、100×100 の大きさに整形し、明るさを調整した後で、反転、回転、80% に縮小したものを生成して、各々別のファイルとして train でディレクトリーに保存します。moto ディレクトリーにあった各々 9 個の画像が、各々 225 個の画像になります。
5. これをもとに例えば「python learn.py -u 50 -e 100 -m dgkw.model -g」と入力して学習プログラムを実行します。そして「python test.py -m dgkw.model」でできたモデルがちゃんと認識するか確認することができます。「-u」や「-e」の値を適切に設定するとリスト 6 のような認識結果も得られます。

リスト 6 水増し画像学習の結果

| | |
|-------------|---|
| 1 d001.jpg | は「だるま型」です。[[0.999844 0.000005 0.000151 0.000000]] |
| 2 d002.jpg | は「だるま型」です。[[0.999749 0.000000 0.000251 0.000000]] |
| 3 d003.jpg | は「だるま型」です。[[0.719442 0.000005 0.280538 0.000016]] |
| 4 g001.jpg | は「画鋏」です。[[0.000000 0.999974 0.000000 0.000026]] |
| 5 g002.jpg | は「画鋏」です。[[0.010267 0.687305 0.203553 0.098875]] |
| 6 g003.jpg | は「画鋏」です。[[0.000000 0.993213 0.005990 0.000796]] |
| 7 k001.jpg | は「割りピン」です。[[0.000057 0.000892 0.294389 0.704662]] |
| 8 k002.jpg | は「クリップ」です。[[0.232162 0.000000 0.767838 0.000000]] |
| 9 k003.jpg | は「クリップ」です。[[0.000000 0.000000 0.999998 0.000002]] |
| 10 w001.jpg | は「割りピン」です。[[0.000000 0.008512 0.028039 0.963449]] |
| 11 w002.jpg | は「割りピン」です。[[0.000000 0.000098 0.000000 0.999902]] |
| 12 w003.jpg | は「クリップ」です。[[0.000001 0.000002 0.998799 0.001198]] |

8.11 YOLO による物体検出

自動車の世界では現在 EV (Electric Vehicle) と自動運転が大きな課題になっています。後者の自動運転で欠かせない技術として高速の物体検出があります。高精度で正確な地図があれば目的地に行くことはできますが、途中の道路上に何があるのかは分かりません。車の運転手は走行中に数秒よそ見をするだけで大事故の危険性が高まります。自動運転車は常に周囲の状況を把握しなければなりません。レーダーを使えば周囲にある物体までの距離が分かります。さらにドップラー効果を利用すれば、その物体が近づいているのか、遠ざかっているのかも分かります。これで衝突は防げるかもしれませんが、レーダーでは信号機が分かりません。赤信号の交差点に突っ込んだら大惨事になります。

画像の中のどこに何があるのかを検出する物体検出は、1970年代から研究されていましたが、なかなかうまく行きませんでした。黒い目と赤い口から顔を検出するぐらいはなんとかできましたが、様々な姿をした人の検出は無理でした。ところが2012年に Alex Krizhevsky らが画像認識 (画像分類) のコンテストである ILSVRC2012 で提出した、畳み込みニューラルネットワーク (CNN) を用いたシステムが優勝したことにより、深層学習を利用した物体検出が主流になりました。そして3年後の ILSVRC2015 で優勝した ResNet は人間よりも少ないエラー率を達成しました。

ここで紹介する YOLO は物体検出のシステムで、最初のバージョンが2016年に発表され現在も改良が続けられています。YOLO は “You only look once” から来ており、候補領域をまず抽出し、その中で検出や分類を行うのではなく、一気に抽出から分類まで行う方式である事を示しています。一気にやることにより精度の点で劣るものの、動画をそのまま処理できる速さを実現しました。自動運転での利用を考えると、人が飛び出したのを数秒後に検出したのでは間に合いません。自動運転で使用される物体検出システムは、YOLO を元に更に改良が進められています。

ここでは2020年6月に Glenn Jocher が発表した YOLO ver.5 (<https://github.com/ultralytics/yolov5>) を実際に使ってみます。

YOLOv5 による物体検出

YOLO ver.5 は python で書かれた PyTorch という名前のフレームワークで動きます。そのため YOLOv5 を動かすためには PyTorch を初め多数の関連ライブラリをインストールする必要がありますが、全て mars にインストール済み^{*27}なので、YOLOv5 本体のみインストールすれば使うことができます。

1. 「教材フォルダ」から「yolov5」フォルダを丸ごと mars のデスクトップにコピーします。
2. コピーした「yolov5」フォルダをダブルクリックして開き、「QTerminal」を起動します。
3. 「source .venv/bin/activate」と入力して仮想環境を起動します。
4. 次のコマンドを入力するとサンプルデータの処理を行います。(本来は「-device 1」は不要ですが新しい GPU に対応できなかったため、古い GPU を使用するこの指定を入れています。)

```
python detect.py --device 1 --source data/images/bus.jpg
```

するとリスト 7 のようなメッセージが表示されます。9 行に「4 persons, 1 bus, 13.7ms」とありますが、画像の中に 4 人とバスを 1 台 13.7 ミリ秒で見つけたという意味です。

^{*27} 大抵の YOLOv5 の解説では、関連ライブラリのインストールから説明されています。PyTorch などはかなり大きなライブラリなので全て入れると 4GB 程度のディスク容量を必要とします。

リスト 7 YOLOv5 による画像認識

```

1 miki@mars:~/Desktop/yolov5$ python detect.py --source data/images/bus.jpg
2 detect: weights=yolov5s.pt, source=data/images/bus.jpg, data=data/coco128.yaml,
  imgs=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=1,
  view_img=False, save_txt=False, save_conf=False, save_crop=False, nosave=False
  , classes=None, agnostic_nms=False, augment=False, visualize=False, update=
  False, project=runs/detect, name=exp, exist_ok=False, line_thickness=3,
  hide_labels=False, hide_conf=False, half=False, dnn=False, vid_stride=1
3 YOLOv5 ^f0^9f^9a^80 v6.2-266-g72cad39 Python-3.10.19 torch-2.5.1+cu124 CUDA
  :1 (NVIDIA GeForce RTX 4060 Ti, 15945MiB)
4
5 ...中略...
6
7 Fusing layers...
8 YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
9 image 1/1 /home/miki/Desktop/yolov5/data/images/bus.jpg: 640x480 4 persons, 1 bus
  , 13.7ms
10 Speed: 0.2ms pre-process, 13.7ms inference, 43.6ms NMS per image at shape (1, 3,
  640, 640)
11 Results saved to runs/detect/exp

```

5. リスト 7 の最後の行に「runs/detect/exp」に結果を保存したとあるので、「runs」フォルダ、「detect」フォルダ、「exp」フォルダの順にダブルクリックをして開いていくと、bus.jpg があります。これをダブルクリックすると図 8.2 のような、検出した物体が枠線で囲われているものが表示されます。枠線に「bus 0.85」のように表示されていますが、これは物体が「bus」で確からしさが 0.85 という意味です。

YOLOv5 用学習データの作成

YOLO に自分が検出して欲しい物体を学習させることができます。そのためにはまず学習データを作成する必要があります。4 つの文具を認識させる場合は、画像ごとに何の画像かが分かるようにしていました。YOLO は物体の位置も答えてくれるので、学習データも「何がどこにあるか」が分かるようにする必要があります。このような学習するための情報をアノテーションと呼びます。

YOLO の場合、画像 (~.jpg) ごとにアノテーションの入ったテキストファイル (~.txt) を用意します。アノテーションの形式は 1 行が 1 つの物体に対応しているので、一つの画像に 5 つの物体があれば、5 行になります。1 行には、物体のクラス番号、物体の中心の X 座標、物体の中心の Y 座標、幅、高さの 5 つが空白で区切られて並びます。座標や幅、高さは、画素数ではなく、画像全体の幅や高さを 1 としたときの相対的な数値になっています。このような情報を作成するためのツールプログラムがいくつかあり、mars には labelImg^{*28} をインストールしたので、これの使い方を説明します。

1. 「yolov5」を開いたところで「QTerminal」を起動します。
2. 次のコマンドを入力して LabelImg を起動します。ただしこれは既にクラス名の入ったファイル (classes.txt) がある場合のコマンドで、そのようなものがない場合は「labelImg」のみで起動します。

```
labelImg data/train/images data/train/labels/classes.txt data/train/labels
```

*28 <https://github.com/heartexlabs/labelImg>

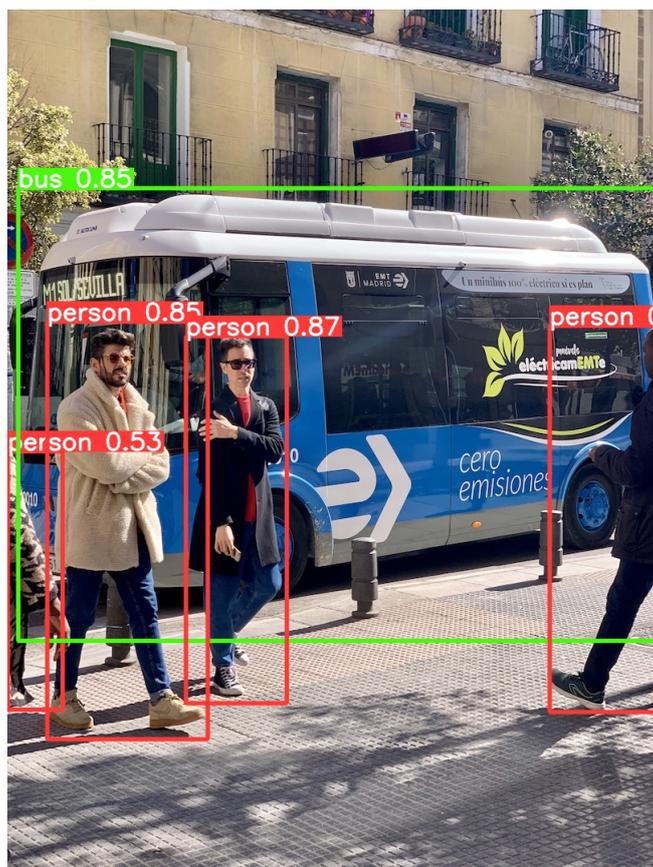


図 8.2 YOLOv5 による画像認識結果

3. 起動すると図 8.3 のような画面になるので、このように左側のところに「PascalVOC」が出ていたら、「ファイル」メニューの中の「PascalVOC」をクリックして「YOLO」に切り替えます。また「表示」メニューの中の「自動で保存する」にチェックを付けておくと、別の画像に移動する際に保存をしてくれるようになります。中央の細い長方形のところに画像が表示されるので、ウィンドウを最大化しておくといいでしょう。
4. 学習用の画像は「yolov5」の中の「data」の中の「train」の中の「images」に入れておきます。5つのサンプル画像が入っているはずですが、学習用の画像を追加するならば同じところに入れます。この入れる場所は後に出てくる設定ファイルに合っていれば、他の場所でも構いません。train」の中にはもう一つ「labels」があります。ここにはサンプル画像の一つのアノテーション情報の入ったファイルとクラス名の入った「classes.txt」というファイルがあります。分担してアノテーション追加の作業を行う場合は、まず全てのクラス名の入った「classes.txt」を作成して、それをコピーして使うのが良いでしょう。
5. 「labelImg」コマンド単独で起動した場合は、左側にある「ディレクトリを開く」をクリックして、画像があるディレクトリ (images) を開いて行って、画像のファイル名が並んでいるところまで行ったら「Open」をクリックします。
6. 「labelImg」コマンド単独で起動した場合は、左側にある「保存先を変更する」をクリックして、アノテーションを入れるディレクトリ (labels) を指定します。YOLO の場合、画像が入っているディレクトリの隣にこのディレクトリを置きます。
7. 中央に最初の画像が表示されます。そこでアノテーションを付ける物体に対して次の操作を繰り返します。
 - (1) **W**を押します。
 - (2) マウスポインターの形が変わって、右と下へ黒い細線が伸びます。物体の上と左がちょうど入ると



図 8.3 LabelImg の最初の画面 (labelImg 単独で起動した場合)

ここまでマウスポインターを移動します。

- (3) ドラッグしてマウスポインターを物体をちょうど囲う右下へ移動します。
- (4) マウスボタンを離すと、クラスを尋ねてくるので新しいクラスの名前を入力するか、既に存在するクラスをリストの中から選択します。

全ての物体に操作を繰り返すと図 8.4 のようになります。



図 8.4 アノテーションの設定が終わったところ

8. まだアノテーションが付いていない画像がある場合は、左側にある「次の画像」をクリックします。

画像の置き場所などが変わると、これまでのアノテーションの結果が表示されなくなることがあります。そのような場合は「ファイル」メニューの「全て初期化する」を選択して、再度設定をやり直すと治るようです。また設定途中で「Attention」のダイアログが出ることがあります。何か設定を変えた場合は「Yes」、そうでない場合は「No」と答えておけば良いでしょう。

学習データによる学習

YOLOv5 で学習させる場合は、まずモデルを選択します。「YOLOv5n」、「YOLOv5s」、「YOLOv5m」、「YOLOv5l」、「YOLOv5x」と5つのモデルがあり、「YOLOv5n」が一番小サイズで、短い時間で認識しますが、精度がその分悪くなります。「YOLOv5x」が一番大きなサイズで、「YOLOv5n」と比べて認識に2倍弱の時間がかかりますが、精度はその分良くなります。他の3つのモデルはその中間の性能になります。特に指定をしない場合は、「YOLOv5s」になります。違うモデルを使用する場合は、後に出てくるコマンドの最後に「--weights yolov5x.pt」のような指定を追加します。

画像がどのディレクトリにあるか、クラスの名前などの設定ファイルを用意します。既に用意したもの(dgkw.yaml)が「data」の中にあるのでこれを利用します。その内容はリスト8のようになっています。

リスト 8 学習のための設定ファイルの内容

```
1 # sample 2022/12/02 by K.Miki
2
3 path: ./ # dataset root dir
4 train: train/images # train images
5 val:   train/images # val images
6 test:  # test images (optional)
7
8 # Classes
9 names:
10  0: 画鋏
11  1: 割りピン
12  2: だるま
13  3: クリップ
```

4行目で学習用データがあるディレクトリを、5行目で検証用データがあるディレクトリを指定しています。今回の設定では学習用データをそのまま検証用で使用しています。10行目以降は、「labels」ディレクトリにあったclasses.txtの内容にゼロからの番号を付けたものになっています。各業の先頭は同じ数の空白が必要なようです。この例のように日本語の名前を付けると、各種グラフで文字化けします。

次のコマンドで学習を実行します。

```
python train.py --batch 10 --epochs 100 --data data/dgkw.yaml --device 1
```

大量のメッセージが表示されます。特に指定をしないとGPUを使用します。「--batch 10」は学習データをまとめる単位で、数字が大きくなるほど認識率が高まりますが、処理時間とGPUのメモリの使用量が増えます。メッセージの中の「CPU_mem」の下に「1.05G」と出ていますが、これがグラフィックボードのメモリーを越えると動きません。marsのグラフィックボードには12GBのメモリーが付いているので、まだかなり余裕があります。「--epochs 100」は何回学習を行うかを示しています。この数字が大きいほど学習は進みますが学習時間がかかり、ある程度の数を越えるとあまり結果が変わらなくなってきます。

学習の進み具合は「runs」の中の「train」の中の「exp」（番号付き）の中の画像などを見ることにより分かります。「results.png」がグラフをまとめた画像になっていますが、グラフがまだまだ上がりそうや下がりそうな場合は epochs の数値を大きくすればよいでしょう。

学習結果の確認

次のコマンドで学習結果を用いてテスト画像を認識させることができます。（1行で入れてください。）

```
python detect.py --source data/train/tests --weights runs/train/exp/weights/best.pt
--device 1
```

「/exp/」の部分は学習を実行した際の最後に表示される「Result saved to」の行の最後の部分と同じものにします。「--source」で認識する画像ファイルそのものか、画像ファイルが入っているディレクトリを指定します。「--weights」で認識に使用する学習結果を指定します。同じディレクトリに best.pt と last.pt の2つのファイルがあります。best.pt は学習中で一番良かった学習結果で、last.pt は最後の学習結果です。内容が同じものになることもあります。

色々なメッセージが表示されますが、最後の行に「Results saved to runs/detect/exp3」と出たならば、「runs」の中の「detect」の中の「exp3」の中に認識結果が入っています。信頼度の低い結果まで描き込まれるので、一つの物体に検索結果が3つも重なっているようなものも出てきます。前にやった一つの画像の一つしか出てこないものの認識であれば、一番確からしいものを認識結果として採用すればよいのですが、こちらは位置が微妙にずれていますので、そう言うわけには行きません。代わりに「--conf-thres 0.3」のような設定を行の最後に追加して実行すると、信頼度が 0.3 より低い結果は表示されなくなり見やすくなります。