第 1 回 概要説明

1.1. はじめに

1.1.1. この講義の目的

この講義では、3 次元コンピュータグラフィックス（3DCG）の技術的基礎について学びます。この目的は、Maya や Blender などの CG ソフトやゲームソフトが、どうやって映像を作っているのか理解することにあります。それにより、CG ソフトを使いこなすのに必要な用語や概念を知るとともに、CG ソフトやゲームソフトを開発するための基礎知識と、基本的なグラフィックスプログラミング技法の習得を目標にします。したがって、この講義の単位取得の条件は CG の理論をもとに自分で考えてグラフィックスプログラミングができるようになることです。

プログラミングとは課題を解決する手順や方法を考えて、それをコンピュータが取り扱い可能な形式で記述することです（図 1.1）。ですから、いくら教科書を読んでプログラミング言語の文法を覚えても、それだけではプログラムが書けるようにはなりません。自分でプログラムが書けるようになるには、実際にプログラムを書く（「コーディングする」と言います）前に、与えられた課題を分析して何を解かなければいけないかを明らかにし、それを問題自体から切り離して抽象化する必要があります。

例えば「1 個 200 円（税込み）のジョナゴールドを 4 個買ったときの代金はいくらか」という課題は、ジョナゴールドの単価を x、個数を y とし

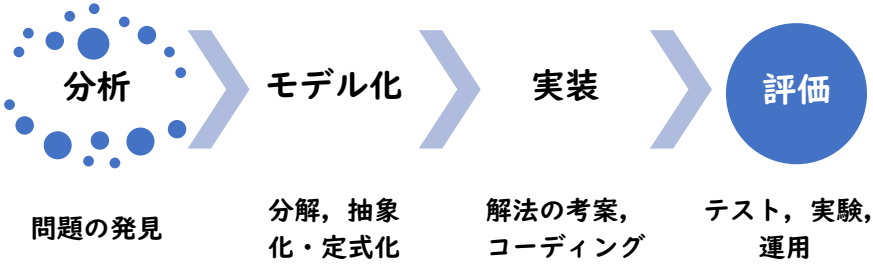


図 1.1 プログラミング的思考

て、代金 z を $z=x\times y$ により求める、というように抽象化できます。また、この支払いに 1,000 円札を使用したとすれば、これを p として、釣銭 c を $c=p-z$ で求めることができます。これは単価と個数から代金を求めるという知識や、代金と支払額から釣銭を求めるという知識を、それぞれ積や差という数式を用いてモデル化したことになります。

さらに、これは $c=p-x\times y$ と書き換えることができます。このようにモデル化によって課題を抽象化することにより、より複雑な課題であっても、単純なモデルの組み合わせでモデル化することが可能になります。

この講義で解説する 3DCG の理論は、実際にゲームやグラフィックスに関連した仕事に就かない限り、使うところはあまりないかも知れません。しかし 3DCG は、物理現象を題材にして課題をモデル化し、プログラムとして実装する訓練には、格好の素材だと考えています。

1.1.2. 成績評価

- 講義中に出す課題：　５点×６回＝３０点
- 宿題：　１０点×７回＝７０点
  - 提出せず：　０点
  - 期限遅れ：　３点
  - 期限内に提出：　６点
  - 指示通り動作：　１０点
- 合計：　１００点

1.1.3. 講義日程

- 第１回　10 月 12 日（火）第３・４時限　概要説明
- 第２回　10 月 19 日（火）第３・４時限　図形の描画
- 第３回　10 月 26 日（火）第３・４時限　座標系と変換の理論
- 第４回　11 月２日（火）第３・４時限　形状の表現と操作
- 第５回　11 月９日（火）第３・４時限　陰影付け
- 第６回　11 月 16 日（火）第３・４時限　テクスチャマッピング
- 第７回　11 月 30 日（火）第３・４時限　マウスとキーボード
- 第８回　12 月２日（木）第３・４時限　アニメーション

1.2. コンピュータグラフィックスについて

1.2.1. コンピュータグラフィックスという用語の意味

コンピュータグラフィックス（Computer Graphics，以下 CG）という用語は、それを使う人や状況によって、次の 2 通りの意味で用いられます。

- コンピュータを使って制作された画像や映像
- CG を制作するための理論や技術

そのため、前者は「CG 画像」や「CG 映像」、あるいは「CG 作品」のように呼ばれることがあります。一方、後者は「CG 理論」「CG 技術」と呼ばれることがあります。

一般に「CG を作る」というと、Autodesk 社の Maya やオープンソースで開発されている Blender のような CG 制作ソフトウェア[1]を使用して CG 画像やムービーを制作する意味のようにとらえられます。しかし、例えばゲームの映像もコンピュータによって生成されています。ゲームの映像はゲームソフト自体か、あるいはそれが使用しているゲームエンジンのライブラリの機能によって、ゲームのプレイ中に動的に作られています。

したがって CG の学習には、CG 制作ソフトウェアの使い方を学んだり、それを利用した表現の作り方を追及したりする制作的な側面と、コンピュータで映像を生成するための理論や、それをもとにしたソフトウェアの開発技術について学ぶという技術的な側面があります。

この講義では後者について学びます。

1.2.2. コンピュータで絵を描くということ

コンピュータで絵を描くということは、画像や図形のデータをコンピュータで処理することにほかなりません。これを人間が絵を描くときに対応付けると、次のようなことを考える必要があります。

- カンバスや絵の具などの画材に相当するものは何か
- 照明やカメラなどの道具はどうやって用意するのか
- 被写体や撮影空間はどのように表現するか

[1] DCC (Digital Contents Creation) ツールと呼ぶことがあります。

したがってコンピュータで映像を生成するには、これらをコンピュータ上で模倣する必要があります。そのために次のような置き換えを行います。

- 形や色、動きなどをデータとして表現する
- 見せ方や描き方をアルゴリズムで表現する

データ構造やアルゴリズムを決定するには、対象を数式で表現することも必要になります。これを、この講義ではモデル化と呼んでいます。

1.2.3. モデル化と CG

今のコンピュータ、というかパソコンは、動画を見たり Web で検索したり、ワープロで文章を書いたり表計算で貸借対照表を作ったりと、様々なことができます。しかし実際には、コンピュータ自体が本来できることは非常に限られています。

- 扱えるものは「数値」と数値で表した「記号」
- できることは「計算」「判断」「記憶」

つまり、コンピュータでは「やりたいこと」と「やれること」の間に大きな隔たりがあるのです。この隔たりのことをセマンティックギャップと呼ぶことがあります。そこで、このセマンティックギャップを埋めるために、対象をコンピュータで扱える形に置き換える必要があります。

- 数値で表す
- 数式で表す
- 手順で表す

物の大きさや重さは数値で表されますし、自由落下は空気抵抗を無視すれば 2 次方程式で表現することができます。紙の上に線分を描くには、たとえば紙に定規を当てて鉛筆を始点から終点まで動かすという手順を用います。このように様々な対象を数値の集合（データ）や数式、あるいは手順などで表すことができます。それらをモデル化し、それにもとづいてコンピュータ上での実装を考えることが、CG の技術の要になります。

1.3. 2DCG と 3DCG

CG は、用途によっていくつかの種類に分類できます。最も大きな分類は「次元」によるものでしょう。

1.3.1. 2DCG（2 次元 CG）

2DCG は平面上に描かれた、平面的な CG を指します。これは以下のモデル化にもとづきます。

- 図形や画像を表現する空間を 2 次元空間としてモデル化
- 絵の具や絵筆・紙のモデル化
- 作画法のモデル化

1.3.2. 3DCG（3 次元 CG）

3DCG は 3 次元空間のシーンを平面上に投影することにより描かれた CG を指します。これは以下のモデル化にもとづきます。

- 図形や画像を表現する空間を 3 次元空間としてモデル化
- 撮像系（カメラ）のモデル化
- 物の形や色をモデル化
- 光源のモデル化

すなわち 3DCG は、コンピュータ内に構築した仮想的な 3 次元空間の情景を映像化した CG であると言えます。

イラストレータがタブレットを用いて風景を描くときは、対象は当然 3 次元であり、それを 2 次元の画面上に再構成します。しかし、これを 3DCG と呼ぶことはあまりありません。3DCG は対象を 3 次元のモデルとしてコンピュータ上に再現するところに中心的な課題があります。

しかし、3DCG であっても最終的な目的は 2 次元の映像を生成することにあります[2]。プリキュアの ED のように 2 次元アニメの見かけ[3]を 3DCG で再現しているものもあります。この場合のモデル化は、2DCG による表現と 3DCG による表現を組み合わせたものになります。2 次元アニメは人手による表現が行われるところに大きな価値を持ちます。そのため、それと 3DCG を整合させるためには、まだまだ多くの課題があります。近年はこれに機械学習を利用する手法がいくつか提案されています。

---

[2] 立体視や 3D プリンタにより 3 次元形状として出力する場合もあります。

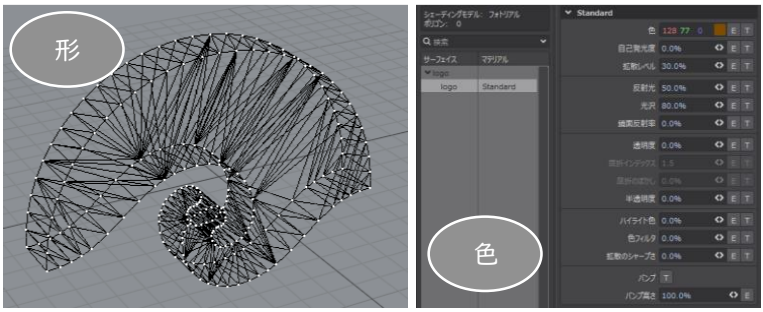
[3] セルルックと言います。

1.3.3. 3DCG の要素

3DCG は、制作的な面でも技術的な面でも、主として次の 2 つの要素に分けることができます。

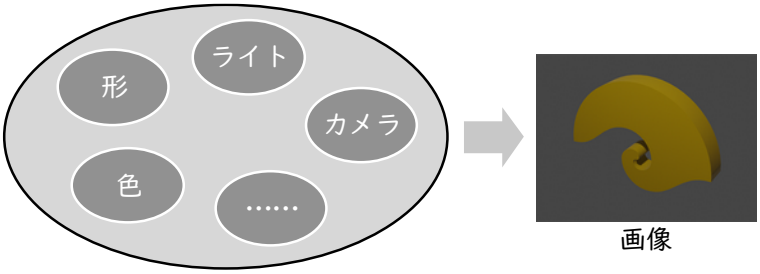
- モデリング＝形や材質特性などの情報をモデル化すること
- レンダリング＝モデル化された情報から映像を生成すること

モデリングとレンダリングは映像制作において異なる段階の作業ですし、ソフトウェアの設計の面でも異なる要素技術を用います。しかし、これらは密接に関連しています。これらのほかにアニメーションも重要な要素ですが、アニメーションはモデリングにもレンダリングにも関わる要素[4]であり、手法によって関わり方も変わるため、この講義では別に扱います。



頂点の位置、頂点の接続関係、等　　反射係数、光沢、等

図 1.2 モデリング



コンピュータ内の仮想的な世界　　画像

図 1.3 レンダリング

[4] 例えば変形アニメーションはモデリングに関連しますし、アニメーションのブレの表現（モーションブラー）はレンダリングに関連します。

(1) モデリング

モデリングは形のデータを作成する作業ですが、色など形に付随するデータも作成します。形を多面体で表現していれば、形のデータは頂点の位置や頂点・稜線・面の接続関係などで構成されます。また色のデータには、物体表面の光の反射率や光沢などがあります（図 1.2）。

(2) レンダリング

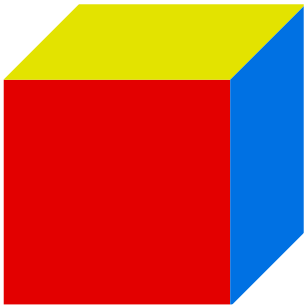
レンダリングはモデリングによって作成された形や色の情報と、カメラ（視点）やライト（光源）の情報をもとにして、それらが表現する世界の見かけの映像を生成する作業です（図 1.3）。コンピュータ内にデータとして保持されているこれらの情報からその見かけの映像を生成する処理は、物理シミュレーションの一種でもあります。そのため、この見かけを精密に再現しようとすれば、それだけ多くの情報や計算時間が必要になります。

1.4. レンダリングの 2 つの目標

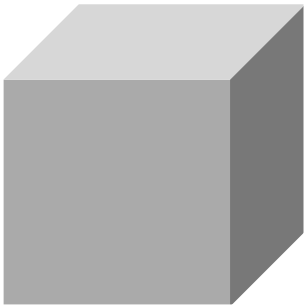
レンダリングには、目的や用途に応じて、2 つの目標があります。

- リアルな映像を生成する
- リアルタイムに映像を生成する

実写映画の特殊効果に使われる CG では、他の実写部分と見分けがつかないリアルな映像が求められます。しかし、そのようなレンダリングには非常に時間がかかります。そのため、映画のように利用者が映像の生成に関与しない用途では、あらかじめ時間をかけて映像を生成する手法が採ら



3 つの四角形の色相が異なる場合



3 つの四角形の明度が異なる場合

図 1.4 色の異なる 3 つの四角形

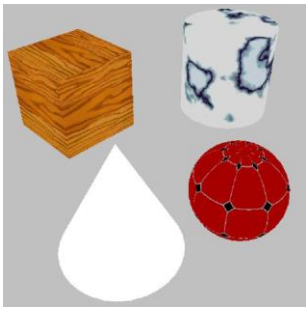
れます。このようなレンダリング方法をプリレンダリングと呼びます。

一方ゲームの映像は、プレイヤーの操作に従ってその場で生成されます。このようなレンダリング方法はリアルタイムレンダリングと呼びます。

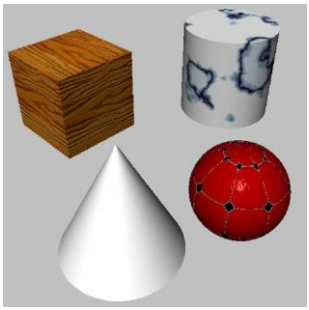
1.4.1. リアルな映像を生成する

CG に求められるリアルさには、形のリアルさや動きのリアルさなど、様々なものがあります。物体に光を当てたときの物体表面の明るさも、見かけのリアルさを決める重要な要素です。例えば、図 1.4 はどちらも色の異なる 3 つの四角形を描いたものです。図 1.4 のどちらがリアルに見えるかは人によりますが、一般に人は物体表面の明度の変化によって物体の立体的な形状を知覚することができます（図 1.5）。この照明による物体表面の明度の分布を、陰影と言います。

人が陰影によって物体の形状を知覚できるのは、反射光の強さが物体表



陰影がない場合



陰影をつけた場合

図 1.5 陰影の有無

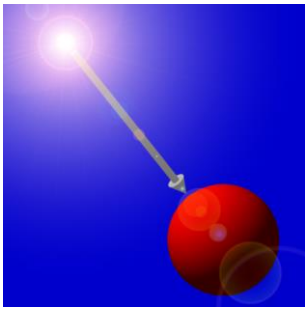


図 1.6 照明と陰影

面の形状と、そこへの入射光との関係で決まることを知っているからです。図 1.6 では、物体表面の光源の方向を向いている部分は明るく、光源と反対側を向いている部分は暗くなっています。人は経験的にそのことを知っているために、陰影によって形を知覚することができます。

余談ですが、図 1.6 では光源の周りにフレアという映像効果を付加しています。これにより、人はこの部分に明るいものがあると認識できます。しかし、その実際の明るさは下地の紙色よりも低くなっています。それなのにこの部分が明るいように思えるのは、これも現実において太陽のような非常に明るい（輝度の高い）光源をカメラで撮影したときに、このような現象が発生することを経験的に知っているからです。

1.4.2. リアルタイムに映像を生成する

ゲームの映像は映画などとは異なり、プレイヤーの操作に合わせた映像を、その場で生成する必要があります。このとき、例えばプレイヤーがコントローラーのスティックを倒してから画面が描き換わるまで何秒もかかってしまうと、おそらくそのゲームでは遊べないでしょう。一般的なディスプレイの表示は 1 秒間に 60 回描き換えられている[5]ので、その時間間隔に間に合わせるなら、1 枚の画像を 1/60 秒≒16.66ms 未満で描き終える必要があります。

このように、リアルタイムレンダリングは制限時間内に映像の生成を完了しなければならないという制約があります。そのため、これに用いられる CG には、プリレンダリングと比較して以下の違いがあります。

- 見かけのリアルさよりも速度を重視する
- シミュレーションは正確さよりも見えの「らしさ」が重要になる

これらによりリアルタイムレンダリングでは、例えばポリゴン（形のデータ）をできるだけ減らしたり[6]、シミュレーションの精度を落としたりします。特にシミュレーションでは正確さよりも「らしさ」が求められるた

[5] 最近はもっと速いもの（1 秒間に 120～240 回）もあります。

[6] ローポリモデリングと呼ばれたりします。

め、例えば陰影の計算方法を簡略化するなど、物理現象をモデル化する際に大幅な近似を行ったり、あらかじめ精密に計算した陰影を用いるなど、事前に計算しておいた結果を用いたりする手法が採られます。

半面、動きのリアルさは重視される傾向にあります。そのため、モーションキャプチャシステム（図 1.7）を用いて現実の動きを取得したり、知性を感じさせる動きを生成するために機械学習が応用されたりしています。

1.4.3. リアルさとリアルタイム性

リアルさを求めようとすれば、その分、計算時間がかかります。一方リアルタイム性を確保しようとすれば、制限時間内に収まらない時間のかかる処理は実行することができません。リアルさ（リアリティ） とリアルタイム性の間には、トレードオフの関係があります。

しかし、リアルタイムレンダリングにおいても、当然リアルな映像が望まれます。リアルタイムレンダリングの目標は、制限時間内に可能な限りリアルな映像を生成することです。同様にプリレンダリングにおいても、コストや納期の制限から映像生成に費やすことのできる時間は有限です。このトレードオフの解消が求められているのです。その結果、この二つのレンダリング方法は、技術的には互いに近づきつつあります。

図 1.7 学術情報センター3 階にあるモーションキャプチャシステム

1.5. 3DCG 映像の生成

3DCG 映像は、次のような手順で生成します（図 1.8）。

- 物体がスクリーンに映る形（投影像）を求める
- その投影像の各部分のディスプレイ上での色を求める
- ディスプレイ全体について色を決定すれば目的の画像が得られる

こうして得られた画像をデジタル[7]画像と呼びます。

1.5.1. デジタル画像

ものの見かけは光の反射光ですが、コンピュータ上でこれを扱うときは、その見かけを空間的に離散化して、光る点の集合として表します。この点

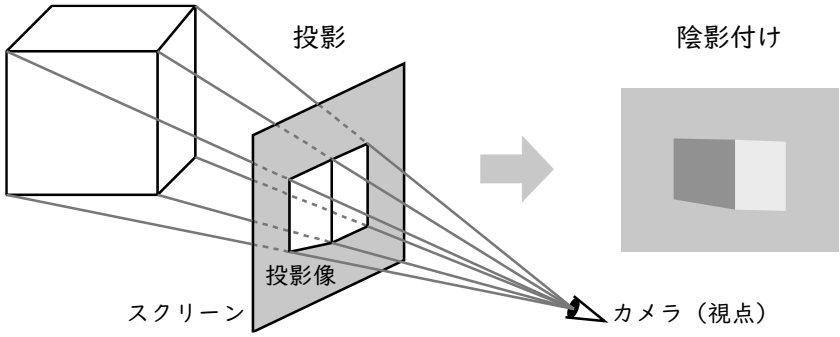


図 1.8 物体のスクリーンへの投影と陰影付け

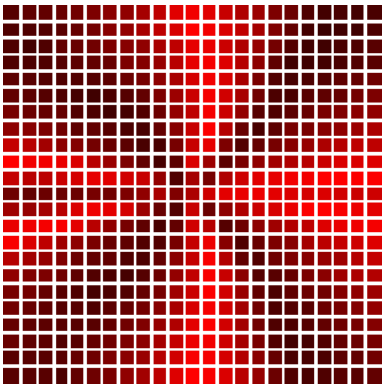


図 1.9 画素の色を変えてグラデーションを表した例

[7] 本当はディジタル (digital) 画像と呼ぶべきですが、この講義では習慣的にデジタル画像と呼んでいます。

のことを画素、あるいはピクセル（pixel）と呼びます。一つ一つの画素は有限の面積を持ち、明るさ、もしくは色の値だけを保持しています。また画素の内部の値は一様ですが、色の違う多くの画素を並べることによって、例えば図 1.9 のようなグラデーションを表現することもできます。デジタル画像の生成は、この一つ一つの画素の色を決める作業です。

1.5.2. 立体図形の投影

物体のスクリーンへの投影像を求めるには、次の 2 通りの方法が用いられます（図 1.10）。

(1) 立体図形上の点のスクリーン上の位置を求める方

この手法は、この講義では投影方式と呼ぶことにします。現在は主にリアルタイムレンダリングで用いられる方法ですが、プリレンダリングでも用いられます。現在のパソコンやゲーム機のグラフィックス表示用ハードウェアは、この手法による映像生成をサポートしています。

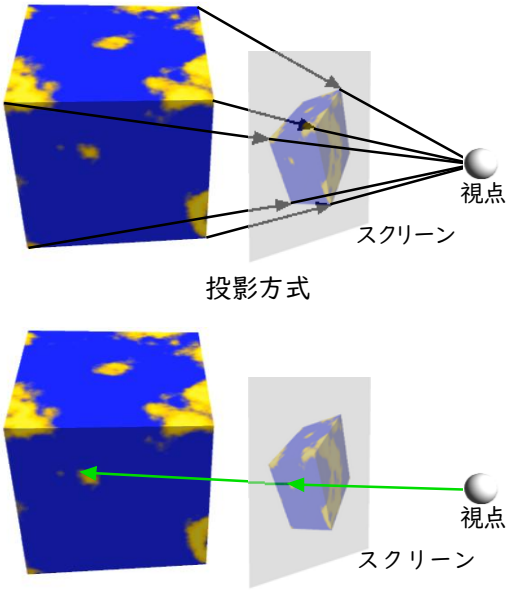


図 1.10 立体図形上の点のスクリーン上の位置を求める方法

(2) スクリーン上の画素を通して何が見えるか調べる方法

この手法は、この講義ではサンプリング方式と呼ぶことにします。主にプリレンダリングで用いられる手法で、レイトレーシング法などが知られています。最近のパソコン用のグラフィック表示用ハードウェアには、この方法をサポートする機能が追加されたものがあります。

1.5.3. 3DCG の処理（投影方式）の流れ

この講義では、主として投影方式の映像生成手法を取り扱います。この方式による映像生成の手順は、例えば図 1.11 のようになります。

この方式では、まず被写体の形の情報を座標変換し、コンピュータ内に想定した 3 次元空間に配置します。それを視点（カメラの位置）から見たものに置き換えた後、スクリーンに投影します。そしてスクリーンへの投影像の画面からはみ出した部分を切り取り、ディスプレイの表示領域に収まるように拡大縮小した後、図形の描画（塗りつぶし）を行います。この塗りつぶしに使う色は、3 次元空間に配置したデータと被写体の色の情報や光源・視点の情報をもとに陰影を計算して決定します。

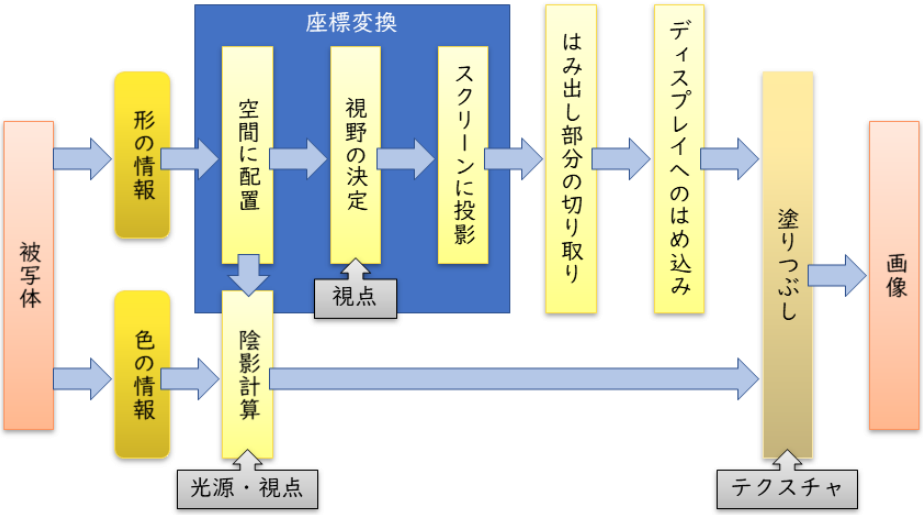


図 1.11 投影方式による映像生成の処理手順

1.5.4. グラフィックスライブラリについて

ソフトウェアは通常プログラミング言語だけで開発できるものではありません。たいていの場合、何らかの「プログラムの部品」を使い、それをプログラミング言語により組み合わせて、目的のプログラムを作成します。例えば C 言語には printf という関数がありますが、printf 自体は C 言語のコンパイラ自体が持っている機能ではありません。これらはソフトウェア開発環境内に別に用意されており、作成したプログラムから呼び出すことにより、リンク時に作成したプログラムに組み込まれます。このように別に用意されているプログラムの部品の集合体のことを、ライブラリと呼びます。printf は C 言語の開発環境に標準的に用意されているので、これが入っているライブラリは C 言語の標準ライブラリと呼ばれます。

もう一つ、プログラムは通常それ単体で動作することもできません。プログラムが動作するには、それを動作させるプラットフォーム、パソコンの場合は Windows や macOS、Linux などのオペレーティングシステム (Operating System, OS) が必要です。プログラムはキーボードやマウスからの入力、画面表示、ネットワーク通信など様々なものとやり取りをしますが、そのためにはパソコンのハードウェアを制御しているオペレーティングシステムの仲立ちが必要になります。プログラムはオペレーティングシステムの機能を呼び出して、これらのハードウェアを制御します。

このオペレーティングシステムの機能の呼び出しも、ライブラリを通じて行います。プログラムはライブラリ関数を呼び出すことによって、オペレーティングシステムの機能を使うことができます。ワープロや表計算など、特定の目的や応用のためのソフトウェアのことをアプリケーションプログラムと言いますが、アプリケーションプログラムがオペレーティングシステムの機能を使うために呼び出すライブラリの関数のことを、特にアプリケーションプログラムインタフェース (Application Program Interface, API) と言います。

ライブラリには目的に応じて様々なものがあります。例えば、機械学習では TensorFlow などがよく使われています。これは作成するプログラム

の構造にも影響を与える（プログラムの書き方が決まってくる）ので、フレームワークと呼ばれることもあります。また画像処理用のライブラリでは、OpenCV が有名です。

グラフィックス表示を行うライブラリは、特にグラフィックスライブラリと呼ばれます。グラフィックス表示はパソコンやスマートフォンなどのハードウェアの機能ですから、グラフィックスライブラリはオペレーティングシステムの機能、すなわち API として提供されています。

そのため、グラフィックスライブラリはプラットフォームごとに独自のものが提供されています。Windows では DirectX、macOS では METAL、Linux では OpenGL が採用されています。このうち OpenGL は異なるプラットフォームで共通に利用可能な API として設計されており、Windows や macOS でも利用することができます。しかし、これは設計が古いために最近のグラフィックスハードウェアの性能を活かしきれないとも言われており、既に macOS では非推奨とされています（使えないことはありません）。なお、OpenGL の後継として Vulkan が開発されています[8]。

また、iOS/iPadOS や Android などのモバイルデバイスでは、OpenGL のサブセットの OpenGL ES を使用することができます。OpenGL ES は Web ブラウザでリアルタイム 3D グラフィックスを実現する WebGL でも採用されています。

これらのグラフィックスライブラリはプラットフォームのグラフィックス機能を呼び出すものであり、プログラムはこれを使ってリアルな 3DCG 表現やアニメーションなどを実現します。しかし、それらは一般に大規模で複雑なプログラムとなり、開発にコストがかかります。そこで、それらの高度なプログラムをライブラリ化して複数のプログラム開発で共通に使用できるようにしたものが openFrameworks や Unity、Unreal Engine

[8] macOS や iOS/iPadOS では、Vulkan は METAL を使って実装されています。

のようなミドルウェア[9]です。これらを使うと、プラットフォームに依存したグラフィックスライブラリを直接使用せずに済むことが多いため、マルチプラットフォームに対応したアプリケーション開発が容易になります。また WebGL では、Three.js や p5.js などのミドルウェアがあります。

1.5.5. CPU と GPU

コンピュータの主要な構成要素（部品）には、CPU（Central Processing Unit，中央演算処理装置）とメインメモリ（Main Memory，主記憶装置）があります。また、パソコンやスマートフォンなどの身の回りにあるコンピュータらしいコンピュータ[10]には、たいていディスプレイ（Display，画面）が付いています。このディスプレイに表示する映像は、図の破線部分のグラフィックスハードウェアによって生成されています（図 1.12）。この部分はビデオカード等の CPU の外部にある拡張ハードウェアの場合もありますが、最近の多くの CPU には内蔵されています。グラフィックスライブラリがオペレーティングシステムを介して制御するのは、主としてこのグラフィックスハードウェアです。

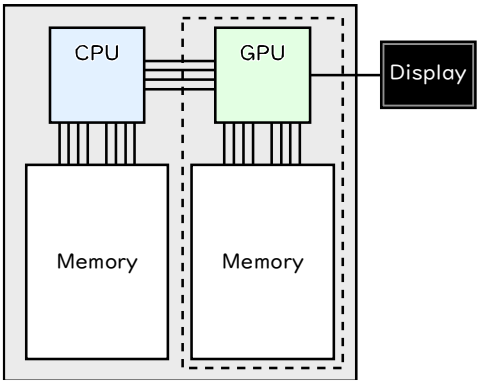


図 1.12 CPU と GPU

[9] Unity や Unreal Engine はゲームの開発を目的としていた（建築ビジュアライゼーションなどゲーム以外の用途にも使えます）ので、ゲームエンジンと呼ばれます。

[10] 最近は炊飯器でも洗濯機でもディスプレイの付いていない組み込みコンピュータが入っているので…

グラフィックスハードウェアの中で図形を描画したり映像を生成したりする中心的な部品は GPU (Graphics Processing Unit, グラフィックス演算処理装置) と呼ばれます。CPU 同様、これにもメモリが接続されています。このメモリはグラフィックスハードウェア上に専用のものが用意されている場合もありますが、グラフィックスハードウェアが CPU に内蔵されている場合は、メインメモリと共用していることもあります。

CPU はメインメモリに格納されているプログラムを取り出し、その命令に従って、やはりメインメモリに格納されているデータを取り出し、それを使って計算した結果をメインメモリに格納します(図 1.13)。CPU とメインメモリを結んでいる線はメモリバスと呼ばれます。

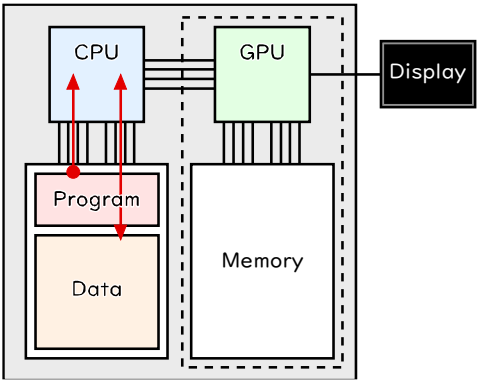


図 1.13 CPU のプログラム実行

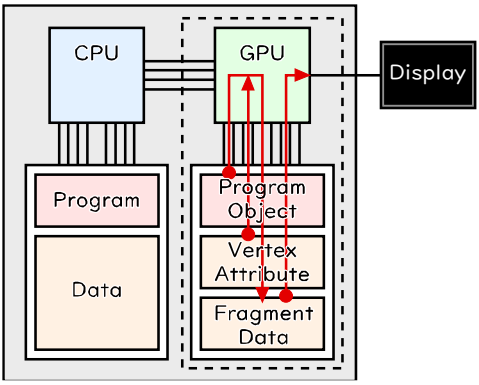


図 1.14 GPU のプログラム実行

GPU も CPU と同様に、それに接続されているメモリに格納されているプログラムオブジェクト (Program Object) を取り出し、その命令に従って、メモリに格納されている頂点属性 (Vertex Attribute, 頂点ごとの位置や色などのデータ) を取り出して、それをもとに図形を描画します。この描画した結果は画素データ (Fragment Data) として、メモリに格納されます。GPU は格納した画素データを周期的に読み出し、それをもとに映像信号を生成して、ディスプレイに図形を表示します（図 1.14)。

CPU のプログラムは、CPU 自身がオペレーティングシステムの指示にしたがって外部記憶装置からメモリに読み込み、実行を開始します。これに対して GPU のプログラムは、CPU のプログラムから転送されたものをグラフィックスハードウェアのメモリに読み込み、CPU の指示によって実行を開始します。したがって GPU は、CPU とは独立して並行動作しますが、CPU の指示に従って補助的に機能します。CPU と GPU を結んでいる線は 入出力バス (Input/Output Bus, I/O バス) と呼ばれます。

このように現代のグラフィックスプログラミングは、CPU と GPU という異なる処理装置の双方に、異なる考え方でプログラミングを行い、それらを組み合わせて使用する、というものになります。

1.6. 準備

1.6.1. 3DCG を理解するための基礎知識

3DCG を学ぶためには、プログラミングやコンピュータシステムに関する知識のほか、数学や物理学の知識も必要になります。とはいえ、最初は高校で習った程度で十分です。

- 数学
  - 線型代数学
  - 解析幾何学
  - 位相幾何学
- 物理学
  - 光学
  - 力学

1.6.2. 形の基本概念

3DCG における形を構成する要素として、ここでは点、線、面、および有向線分とベクトルについて復習します（図 1.15、図 1.16）。

(1) 点

点は位置を表します。唯一、実数で表される情報です（図 1.15 (1)）。

(2) 線

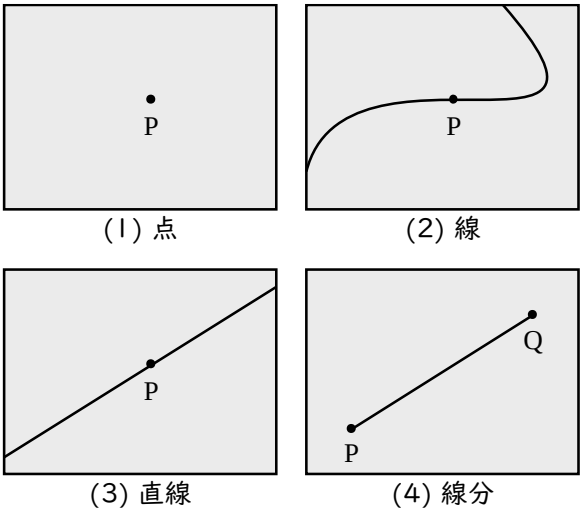
線は点の集合です（図 1.15 (2)）。

(3) 直線

直線は 2 点によってただ 1 つ決定される線です。3 点が一直線上にあれば、3 点のうち 1 点が残りの 2 点の間にあります。また、直線はその上の 1 点によって 2 つに分けられます。直線はその 2 つの方向に限りがありません（図 1.15 (3)）。

(4) 線分

直線上の 2 定点 P と Q の間にある点全体と、点 P および Q により 1 つの線分を形作ります。P と Q は端点と言います（図 1.15 (4)）。



(1) 点　(2) 線

(3) 直線　(4) 線分

図 1.15 点と線

(5) 有向線分

線分の端点の一方が始点であり，もう一方が終点であるものを有向線分と言います。向きを持ちます（図 1.16 (5)）。

(6) ベクトル

ベクトルは有向線分から位置の情報を取り去ったものです。平行移動しても不変な成分を表します（図 1.16 (6)）。

- 方向
- 長さ $|\mathbf{a}|$

零ベクトル $\mathbf{o}$ は長さ $|\mathbf{o}|$ が $|\mathbf{o}| = \mathbf{0}$ のベクトルです。

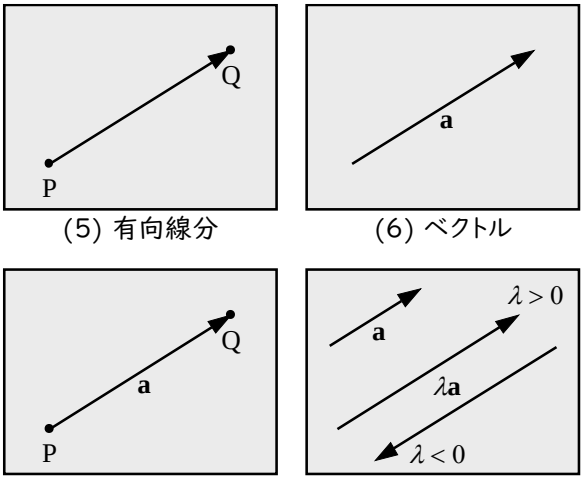
(7) 有向線分のベクトル

始点 P、終点 Q の有向線分のベクトル $\mathbf{a}$ は、次のように表します（図 1.16 (7)）。

$$\mathbf{a} = \overrightarrow{\mathrm{PQ}} \tag{1.1}$$

(8) ベクトルのスカラー倍

ベクトル $\mathbf{a}$ を表す有向線分の長さを λ 倍します（λ は実数）。λ が負（$\lambda < 0$）のときは、向きが反転します（図 1.16 (8)）。



(5) 有向線分　(6) ベクトル

(7) 有向線分のベクトル　(8) ベクトルのスカラー倍

図 1.16 有向線分とベクトル

- $1\mathbf{a} = \mathbf{a}$
- $0\mathbf{a} = \mathbf{o}$
- $-1\mathbf{a}$ ： $\mathbf{a}$ の向きを反転する
- λ ： スカラー

(9) ベクトルの和

二つのベクトル $\mathbf{a}$、$\mathbf{b}$ の和 $\mathbf{a}+\mathbf{b}$ は、$\mathbf{a}$ と $\mathbf{b}$ を合成したものになります（図 1.17）。

(10) 一次結合

次式の $\mathbf{b}$、すなわち有限個のベクトル $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ と同じ数のスカラー $\lambda_1, \lambda_2, \cdots, \lambda_k$ のそれぞれの積の和は、ベクトル $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ の一次結合（または線形結合）と言います。

$$\mathbf{b} = \lambda_1 \mathrm{a}_1 + \lambda_2 \mathrm{a}_2 + \cdots + \lambda_k \mathrm{a}_k \qquad (1.2)$$

(11) 一次独立と一次従属

あるベクトル列 $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ の次の一次結合が零ベクトル $\mathbf{o}$ だったとします。

$$\lambda_1 \mathbf{a}_1 + \lambda_2 \mathbf{a}_2 + \cdots + \lambda_k \mathbf{a}_k = \mathbf{o} \qquad (1.3)$$

この式が成り立つのが $\lambda_1 = \lambda_2 = \cdots = \lambda_k = 0$ のときに限られる場合、このベクトル列は一次独立（または線形独立）と言い、それ以外の場合を一次従属（または線形従属）と言います。

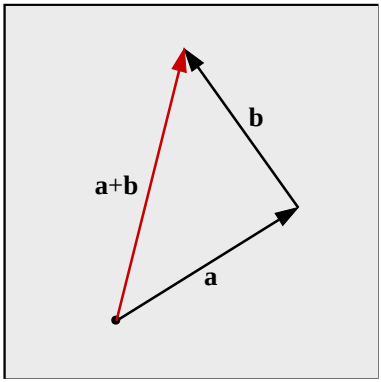


図 1.17 ベクトルの和

(12) 直線座標系

3 次元空間中の点 P の位置は、3 つのベクトル $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ の一次結合で表すことができます。この $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ を点 P が存在する座標系の基底ベクトル（または軸ベクトル）といい、$\lambda_1, \lambda_2, \lambda_3$ を $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ を基底（または軸）とする座標系における座標値と呼びます（図 1.18）。

$$\overrightarrow{\mathrm{OP}} = \lambda_1 \mathbf{a}_1 + \lambda_2 \mathbf{a}_2 + \lambda_3 \mathbf{a}_3 \tag{1.4}$$

(13) 正規直交座標系

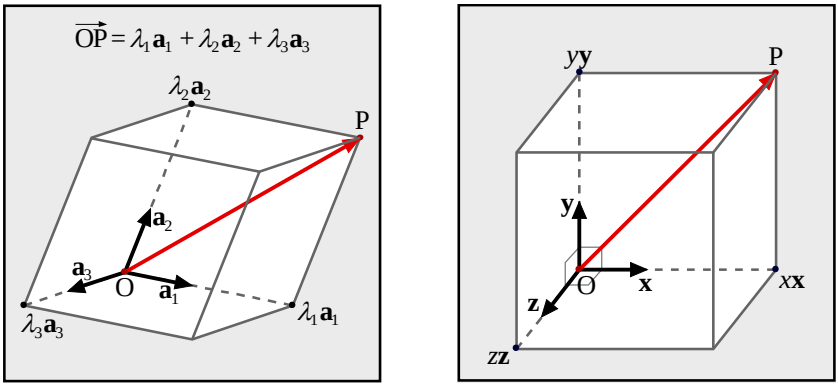
直線座標系において、基底ベクトル $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ の長さが $|\mathbf{a}_1| = |\mathbf{a}_2| = |\mathbf{a}_3| = 1$ であり、それらが互いに直交している場合を、正規直交座標系と言います。

(14) 座標値と位置ベクトル

基底ベクトル $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ をそれぞれ $\mathbf{x} = (1,0,0)$、$\mathbf{y} = (0,1,0)$、$\mathbf{z} = (0,0,1)$ とすれば、これらは互いに直交し、かつ $|\mathbf{x}| = |\mathbf{y}| = |\mathbf{z}| = 1$ となって、正規直交座標系の基底ベクトルの条件を満たします。そこで $\lambda_1, \lambda_2, \lambda_3$ をそれぞれ $\lambda_1 = x$、$\lambda_2 = y$、$\lambda_3 = z$ に置き換えると、$\overrightarrow{\mathrm{OP}}$ は次のように表すことができます。

$$\overrightarrow{\mathrm{OP}} = x\mathbf{x} + y\mathbf{y} + z\mathbf{z} \tag{1.5}$$

すなわち、$\overrightarrow{\mathrm{OP}}$ は座標値が (x, y, z) である点 P の位置ベクトルです。



直線座標系　　正規直交座標系

図 1.18 位置ベクトルと座標系

(15) 右手系と左手系

3 次元空間の場合、2 次元の空間の基底ベクトル $\mathbf{x}$、$\mathbf{y}$ に対する 3 つ目の基底ベクトル $\mathbf{z}$ の向きは、2 通り定義することができます。いま、自分の目の前で $\mathbf{x}$ 方向を右、$\mathbf{y}$ 方向を上としたとき、$\mathbf{z}$ が自分の方を向く場合を右手系、自分に対して反対側を向く場合を左手系と言います。

図 1.19 のように、右手系は右手の親指を $\mathbf{x}$、人差し指を $\mathbf{y}$、中指を $\mathbf{z}$ として互いに直交するように伸ばしたときの手の形に相当し、左手系は左手の親指を $\mathbf{x}$、人差し指を $\mathbf{y}$、中指を $\mathbf{z}$ として互いに直交するように伸ばしたときの手の形に相当します。この講義では右手系を採用します。

(16) 線分の方程式

2 点 P_0、P_1 を端点とする線分について考えます。この線分を点 P_0 から点 P_1 に向かう有向線分とし、$\mathbf{p}_0$、$\mathbf{p}_1$ をそれぞれ点 P_0、点 P_1 の位置ベクトルとするとき、この有向線分のベクトル $\mathbf{v}$ は、次のようになります。

$$\mathbf{v} = \overrightarrow{\mathrm{P}_0\mathrm{P}_1} = \mathbf{p}_1 - \mathbf{p}_0 \tag{1.6}$$

この $\mathbf{v}$ を $\mathbf{v} = (l, m, n)$、P_0 の位置を $\mathbf{p}_0 = (x_0, y_0, z_0)$、線分上の点 P の位置を $\mathbf{p} = (x, y, z)$ とするとき、この線分の方程式は次のようになります。

$$\frac{x - x_0}{l} = \frac{y - y_0}{m} = \frac{z - z_0}{n} \tag{1.7}$$

しかし、この式には等号 $(=)$ が 2 つあって、プログラムで表現するのが

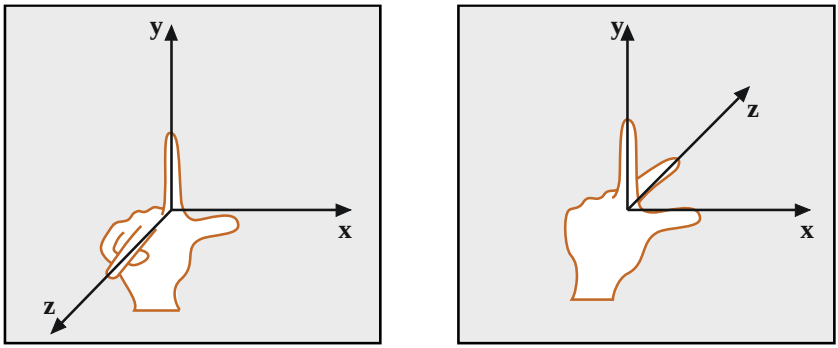


図 1.19 右手系と左手系

面倒になります。そこで、これを媒介変数 t を用いて、2 点 P_0、P_1 を結ぶ線分上の点 P の位置 $\mathbf{p}(t)$ を次のように表す場合があります（図 1.20）。

$$\mathbf{p}(t) = \mathbf{v}t + \mathbf{p}_0 = \mathbf{p}_0(1-t) + \mathbf{p}_1 t \quad (0 \le t \le 1) \tag{1.8}$$

(17) 面

面は線の集合です。線は点の集合ですから、面は点の集合でもあります。

(18) 平面

平面は一直線上にない 3 点によってただ 1 つ決定されます。2 点が平面上にある時、この 2 点を通る直線全体がこの平面上にあります。平面はその上にある直線によって 2 つに分けられます。平面はすべての方向に無限に伸びており、その平面が存在する 3 次元空間を 2 つの部分に分けます。

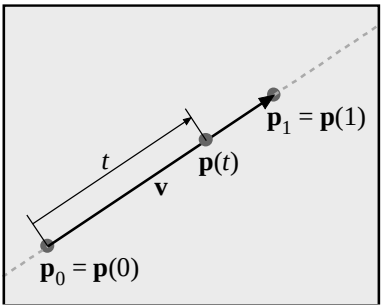


図 1.20 線分の媒介変数表現

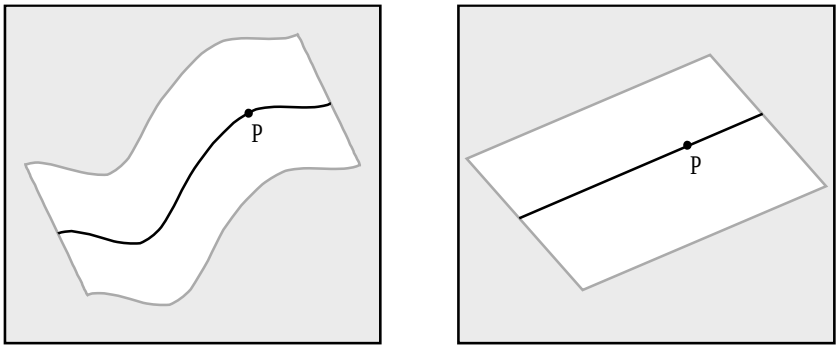


図 1.21 面

(19) 接線ベクトルと法線ベクトル

接線ベクトルは面に接する平面（接平面）上のベクトルです。法線ベクトルはこの平面の法線ベクトルです（図 1.20）。

(20) ベクトルの内積

2 つのベクトル $\mathbf{u}$、$\mathbf{v}$ の内積 $\mathbf{u}\cdot\mathbf{v}$ は、次のように定義されます。

$$\mathbf{u}\cdot\mathbf{v} = |\mathbf{u}||\mathbf{v}|\cos\theta \tag{1.9}$$

ここで "·" はドット演算子と言います。これは $\mathbf{u}$、$\mathbf{v}$ の 2 つのベクトルのなす角が θ のとき、$\mathbf{u}\cdot\mathbf{v}$ は $\mathbf{v}$ を $\mathbf{u}$ に写像した長さ $|\mathbf{v}|\cos\theta$ と $\mathbf{u}$ の長さ $|\mathbf{u}|$ を 2 辺の長さとする長方形の面積になります（図 1.23）。

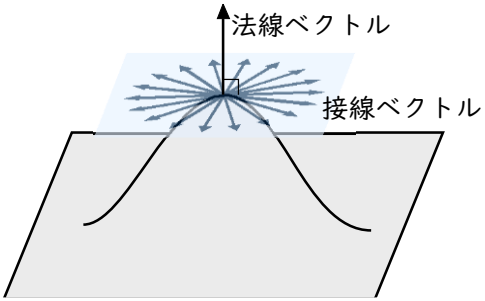


図 1.22 接線ベクトルと法線ベクトル

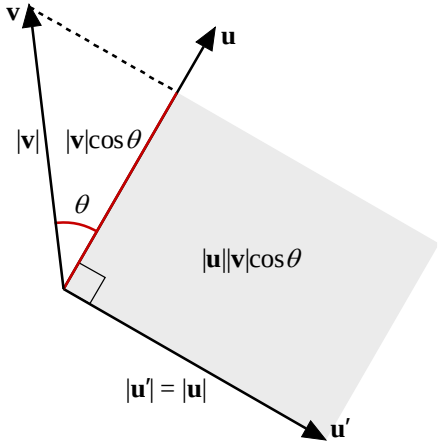


図 1.23 ベクトルの内積

ここで余弦定理 $|\mathbf{u}-\mathbf{v}|^2=|\mathbf{u}|^2+|\mathbf{v}|^2-2|\mathbf{u}||\mathbf{v}|\cos\theta$ より

$$|\mathbf{u}|^2+|\mathbf{v}|^2-|\mathbf{u}-\mathbf{v}|^2=2|\mathbf{u}||\mathbf{v}|\cos\theta=2\mathbf{u}\cdot\mathbf{v} \tag{1.10}$$

ですから、$\mathbf{u}=(u_x,u_y,u_z)$、$\mathbf{v}=(v_x,v_y,v_z)$ とすれば、

$$\begin{aligned}|\mathbf{u}|^2&={u_x}^2+{u_y}^2+{u_z}^2\\ |\mathbf{v}|^2&={v_x}^2+{v_y}^2+{v_z}^2\\ |\mathbf{u}-\mathbf{v}|^2&=(u_x-v_x)^2+(u_y-v_y)^2+(u_z-v_z)^2\end{aligned} \tag{1.11}$$

より

$$|\mathbf{u}|^2+|\mathbf{v}|^2-|\mathbf{u}-\mathbf{v}|^2=2u_xv_x+2u_yv_y+2u_zv_z \tag{1.12}$$

となります。したがって、

$$\mathbf{u}\cdot\mathbf{v}=|\mathbf{u}||\mathbf{v}|\cos\theta=u_xv_x+u_yv_y+u_zv_z \tag{1.13}$$

ですから、2 つのベクトルの内積は、それぞれの要素の積の和になります。

(21) ベクトルの外積

2 つの 3 次元のベクトル $\mathbf{u}$、$\mathbf{v}$ の外積 $\mathbf{u}\times\mathbf{v}$ は、次のように定義されます。なお、3 次元のベクトルの外積はクロス積と呼ばれます。

$$\begin{aligned}\mathbf{u}\times\mathbf{v}&=\left(\begin{vmatrix}u_y & u_z\\ v_y & v_z\end{vmatrix},\ \begin{vmatrix}u_z & u_x\\ v_z & v_x\end{vmatrix},\ \begin{vmatrix}u_x & u_y\\ v_x & v_y\end{vmatrix}\right)\\ &=(u_yv_z-u_zv_y,\quad u_zv_x-u_xv_z,\quad u_xv_y-u_yv_x)\end{aligned} \tag{1.14}$$

このとき、

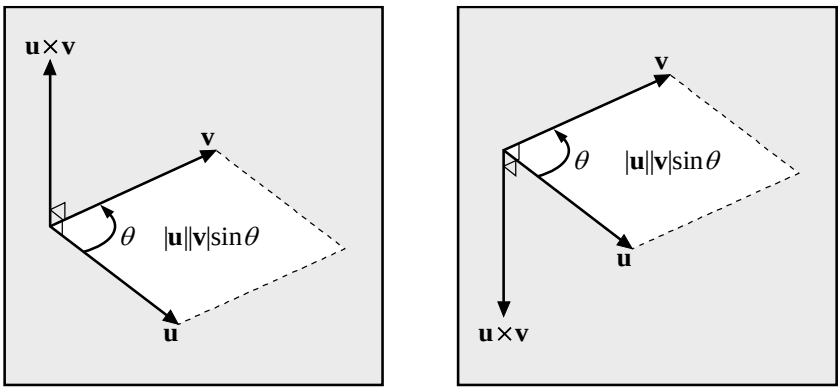


図 1.24 ベクトルの外積

$$
\begin{aligned}
|\mathbf{u}\times\mathbf{v}|^2 &= \left(u_y v_z - u_z v_y\right)^2 + (u_z v_x - u_x v_z)^2 + \left(u_x v_y - u_y v_x\right)^2 \\
&= \left({u_x}^2 + {u_y}^2 + {u_z}^2\right)\left({v_x}^2 + {v_y}^2 + {v_z}^2\right) - \left(u_x v_x + u_y v_y + u_z v_z\right)^2 \\
&= |\mathbf{u}|^2 + |\mathbf{v}|^2 - (\mathrm{u}\cdot\mathrm{v})^2 = |\mathbf{u}|^2|\mathbf{v}|^2 - (|\mathbf{u}||\mathbf{v}|\cos\theta)^2 \\
&= |\mathbf{u}|^2|\mathbf{v}|^2(1-\cos^2\theta) = |\mathbf{u}|^2|\mathbf{v}|^2\sin^2\theta
\end{aligned}
\tag{1.15}
$$

ですから、

$$
|\mathbf{u}\times\mathbf{v}| = |\mathbf{u}||\mathbf{v}|\sin\theta \tag{1.16}
$$

となります。したがって外積の絶対値は、$\mathbf{u}$ と $\mathbf{v}$ が作る平行四辺形の面積になります。

また $\mathbf{u}\times\mathbf{v}$ の向きは、$\mathbf{u}\to\mathbf{v}$ の方向に角度 θ を正に取った場合、右手系の座標系なら $\mathbf{u}\to\mathbf{v}$ の回転に対して右ネジの方向、左手系の座標系なら左ネジの方向になります（図 I.24）。

(22) 三角形と平面の方程式

3 点 p_0, p_1, p_2 を頂点とする三角形について考えます。この三角形の法線ベクトル n は、外積を使って求めることができます（図 I.25）。

$$
\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0)\times(\mathbf{p}_2 - \mathbf{p}_0) \tag{1.17}
$$

いま、点 $p = (x, y, z)$ がこの三角形上にあるとします。$p_0 = (x_0, y_0, z_0)$ としたとき、ベクトル $p - p_0$ もこの三角形上にあります。これはこの三角形の接線ベクトルです。

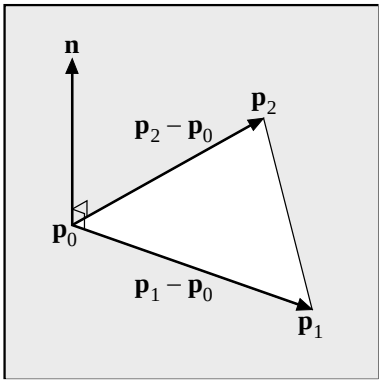


図 I.25 三角形の法線ベクトル

法線ベクトルは接線ベクトルと直交しますから、

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0 \tag{1.18}$$

が成り立ちます。$\mathbf{n} = (a, b, c)$ とすれば、

$$\begin{aligned} (a, b, c) \cdot (x - x_0, y - y_0, z - z_0) &= 0 \\ a(x - x_0) + b(y - y_0) + c(z - z_0) &= 0 \\ ax + by + cz - ax_0 - by_0 - cz_0 &= 0 \end{aligned} \tag{1.19}$$

となります。ここで $d = -ax_0 - by_0 - cz_0$ とおけば、

$$ax + by + cz + d = 0 \tag{1.20}$$

となり、この三角形を含む平面の方程式が得られます。

(23) 三角形の媒介変数方程式による表現

三角形も線分と同様に、媒介変数を使って表すことができます。図 1.26 のように、$\mathbf{p}_0$ を原点として $\mathbf{p}_1$ に向かう有向線分上の点の位置を、u を媒介変数に用いて

$$\mathbf{p}_{01}(u) = (\mathbf{p}_1 - \mathbf{p}_0)u \tag{1.21}$$

と表します。同様に $\mathbf{p}_0$ を原点として $\mathbf{p}_2$ に向かう有向線分上の点の位置を、v を媒介変数に用いて

$$\mathbf{p}_{02}(v) = (\mathbf{p}_2 - \mathbf{p}_0)v \tag{1.22}$$

と表します。すると $\mathbf{p}_0$ を原点とする、この三角形上の点の位置 $\mathbf{p}(u, v)$ は、この 2 つの位置ベクトル $\mathbf{p}_{01}(u)$、$\mathbf{p}_{02}(v)$ と $\mathbf{p}_0$ を合成したものなので、

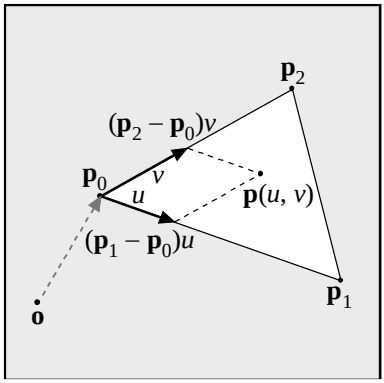


図 1.26 三角形の媒介変数方程式による表現

$$\begin{aligned}\mathbf{p}(u,v) &= \mathbf{p}_{01}(u) + \mathbf{p}_{02}(v) + \boldsymbol{p}_0 \\ &= (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v + \mathbf{p}_0 \\ &= \mathbf{p}_0(1-u-v) + \mathbf{p}_1 u + \mathbf{p}_2 v \end{aligned} \quad (1.23)$$

となります $(u \geq 0, v \geq 0, u+v \leq 1)$。

1.6.3. 開発環境の整備

この講義ではプログラムの説明に Microsoft 社の Visual Studio Code というテキストエディタを使います。Visual Studio Code のインストーラを入手して、使用するパソコンにインストールしてください。

(1) インストーラをダウンロードする

Visual Studio Code は https://code.visualstudio.com/download からダウンロードできます（図 1.27）。使用するプラットフォームに対応したものをダウンロードしてください。それぞれのアイコンの下のボタンをクリックすれば、最適なものがダウンロードされます。

Windows 版の User Installer はユーザごとに個人的な環境下にインス

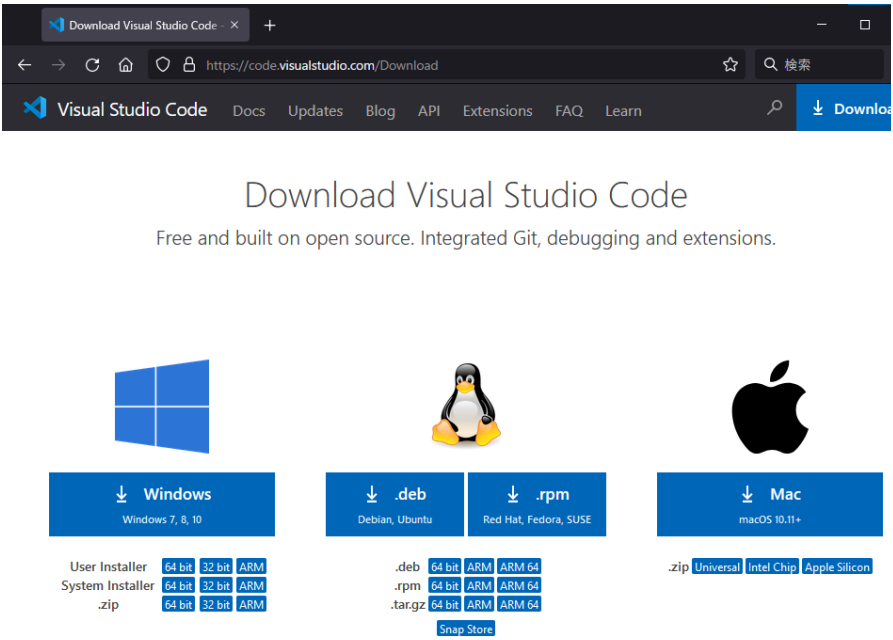


図 1.27 Visual Studio Code のダウンロードページ

トールするので、管理者権限は必要ありません。System Installer は PC のシステム環境下にインストールします。これは PC を複数のユーザで共用しているときに一つの Visual Studio Code を共有することができますが、インストールするには管理者権限が必要になります。

(2) インストールする

インストーラがダウンロードできたら、インストーラを起動してインストールを開始してください。特に設定することはないので、「次へ」のボタンをクリックして、最後に「完了」のボタンをクリックしてください。

(3) 拡張機能をインストールする

Visual Studio Code のインストールが完了したら、そのまま Visual Studio Code を起動してください。次に拡張機能をインストールしますので、拡張機能のペインを開きます（図 1.28)。メニューなどは既に日本語化されていると思いますが、もし英語のままなら“japanese”で検索するなどして Japanese Language Pack for Visual Studio Code の拡張機能をインストールしてください。

次に Live Server をインストールしてください（図 1.29)。Live Server

図 1.28 Japanese Language Pack のインストールの確認

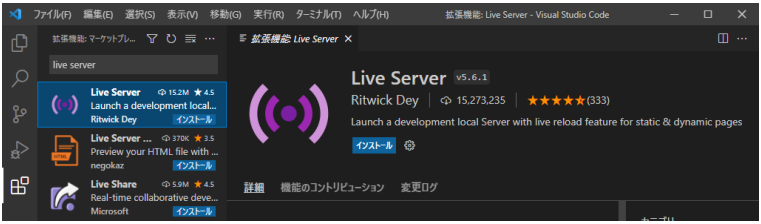


図 1.29 Live Server のインストール

は Visual Studio Code 内で Web サーバを起動することにより、編集中の Web ページを別のブラウザでプレビューすることができます。

(4) 保存先のフォルダを作成する

作成したファイルを保存するフォルダを作ってください（図 1.30）。Live Server はフォルダ単位でファイルを参照します。作成する場所はどこでも構いませんが、Web サーバから参照されるので、フォルダ名は半角英数字にした方が無難でしょう。

(5) 保存先のフォルダを開く

作成したフォルダを Visual Studio Code で開いてください（図 1.31、図 1.32）。するとフォルダの作成者を信頼するかどうかを聞いてきますから、フォルダの作成者である自分を信頼してください（図 1.33）。

(6) ファイルを作成する

フォルダが開けたら Ctrl-N をタイプして、そのフォルダにファイルを新規に作成します（図 1.34）。その後すぐに Ctrl-S をタイプして、そのファイルを保存します（図 1.35）。このときファイル名の拡張子を **.html** にすれば（図 1.36）、「ファイルの種類」は設定しなくても HTML ファイ

図 1.30 保存先のフォルダの作成

図 1.31 フォルダを開く

ルの編集モードになります（図 1.37）。

HTML ファイルが作成できたら、一旦 Visual Studio Code でフォルダを閉じる（Ctrl-K F）か、Visual Studio Code 自体を一度終了して、もう一度保存先のフォルダを開いてください。これで Visual Studio Code の

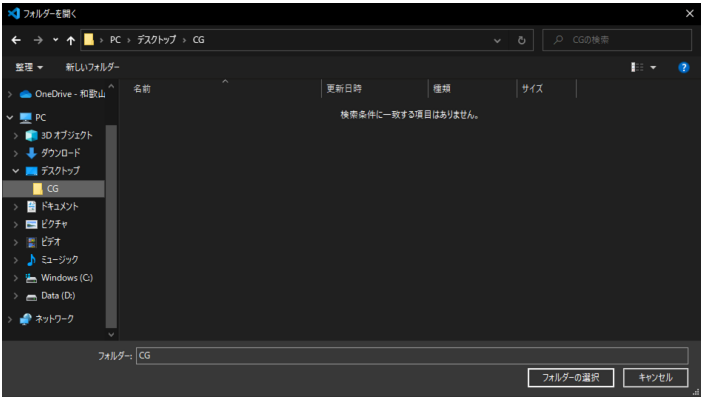


図 1.32 保存先のフォルダを開く

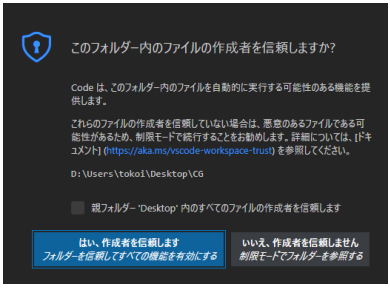


図 1.33 フォルダの作成者を信頼する

図 1.34 新規ファイルの作成

ウィンドウの右下に「Go Live」という文字が現れます[11]。

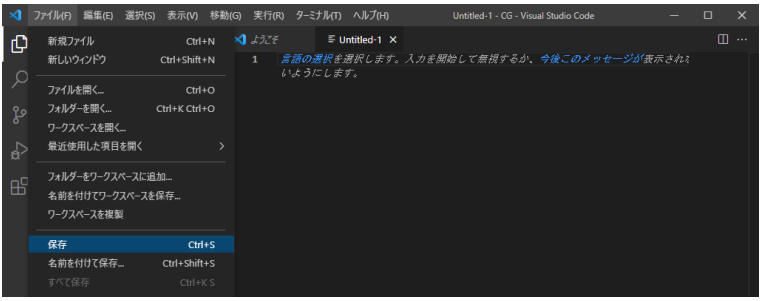


図 1.35 ファイルの保存

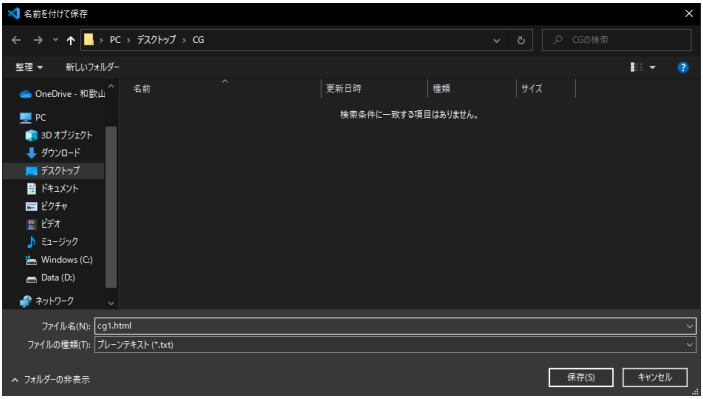


図 1.36 保存するファイル名は cg1.html にする

図 1.37 HTML ファイルの作成

[11] Live Server は HTML ファイルの入っていないフォルダを開いても起動しないので、フォルダを開いた後に HTML ファイルを作った場合はフォルダを開き直します。

1.7. Web ページの作成

1.7.1. ベースの HTML ファイルの作成

最初に、次の HTML テキストを入力してください。この HTML テキストは Web ページ内に図形描画を行う canvas 要素を一つだけ作ります。これから、ここに図形を描画するスクリプトを作成します。

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>CG1</title>
</head>
<body>
  <canvas id="CG1" width="400" height="300">
    HTML5 の canvas が使えません
  </canvas>
</body>
</html>
```

これを保存します。既に cg1.html というファイル名が付いているはずですから、Ctrl-S をタイプすれば、そのファイルに上書き保存されます。

次に、Visual Studio Code のウィンドウの右下の「Go Live」という文字をクリックしてください。すると Web ブラウザが起動して、Visual Studio Code の中で起動している Web サーバに接続して、編集中の Web ページが表示されます。

ただし、このままだと canvas 要素とブラウザの背景が同じ色になっているために見分けが付きません。そこで、例えば次の**太字**の内容を追加することによって、canvas 要素に枠を付けることができます。

```
  <canvas id="CG1" width="800" height="600" style="border: inset;">
```

「style="border: inset;"」の inset は outset あるいは solid、dashed などでも構いません。これを追加した後 Ctrl-S をタイプしてファイルを上書き保存すれば、Web ブラウザの表示が更新されて canvas 要素

に枠が付きます。

1.7.2. 2D グラフィックス

試しに、この canvas 要素に 2D グラフィックスを描いてみましょう。そのためのスクリプトを、<canvas>～</canvas> の後に追加します。

```
<body>
  <canvas id="CG1" width="400" height="300" style="border: inset;">
    HTML5 の canvas が使えません
  </canvas>
  <script>
    const canvas = document.getElementById('CG1')
    const context = canvas.getContext('2d')

    context.beginPath()
    context.moveTo(10, 20)
    context.lineTo(60, 20)
    context.lineTo(60, 70)
    context.lineTo(10, 70)
    context.stroke()
  </script>
</body>
```

これは図 1.38 の図形を描きます。document.getElementById('CG1') は document から id プロパティが 'CG1' の要素を探して返します。この場合は上の canvas 要素が返されます。canvas.getContext('2d') は、この canvas 要素から 2D グラフィックスのコンテキスト（グラフィックス描画の状態を保持する作業用のメモリなどの集合体）を返します。

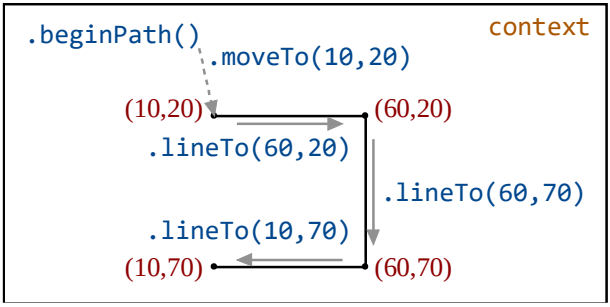


図 1.38 折れ線の描画

context.beginPath() は、この canvas 要素のコンテキストが保持するパス（線）のリストを空にします。次に context.moveTo(10, 20) により仮想的な「ペン」を (10,20) の位置に移動します。そして、そこから context.lineTo(60, 20) として (60,20) に線を引きます。こうして、このスクリプトは 4 点 (10,20)、(60,20)、(60,70)、(10,70) を結ぶ折れ線を描きます。この結果は図 1.39 のようになります。

なお、最後（context.stroke() の前）に context.clothPath() を置くと、終点から始点に向かって線分を描いて図形を閉じることができます。

図 1.39 描画される折れ線