はじめに

コンピュータグラフィックス(Computer Graphics，以下 CG)は昨今の映画やゲームなどに見られるデジタル映像としてご存知の方も多いと思います．さまざまな情報を可視化するという意味でいえば，CG は今日の基本メディアといえるでしょう．一方で，「CG と数学にはどんな関係があるのか」については想像しにくいかもしれません．そもそも CG の研究とは何だろうか？ と思われる方も多いでしょう．

本書は，数学と CG の両方に興味がある意欲的な読者を想定し，CG では何を研究して，どんな数学が駆使され，そしてどのような数理モデルが作られていくのかを，たくさんの映像事例とともに紹介します．多様な CG 表現を支える数学的概念や手法のことを，本書では映像数学と総称します．

本書を読むにあたっての予備知識としては大学理工系の 1, 2 年生の知識が望ましいですが，それより大切なのは CG や数学への好奇心です．CG 映像には馴染みがあっても，作り方はよくわからない人も多いでしょうから，本書では CG 制作の基礎から説明します．一方数学については厳密な定義や証明は出てきません．技術的細部に捉われることなく，読み進めてください．CG と数学の意外に(?)深いつながりと，それがもたらす映像のインパクトをぜひ楽しんでいただきたいと思います．

ここで CG の歴史をごく大雑把に振り返ってみましょう．それによって CG の研究とは何かが見えてくると思うからです．CG という言葉が一般的になったのは近年のことですが，1950 年代からコンピュータを用いて機械設計や形状デザインの効率化を目指す技術開発が進みました．これらの技術群は CAD (Computer Aided Design)と呼ばれ，現在も進化し続けています．1960 年代には，その設計対象の表示技術として CG が存在しました．形や色の情報などが，すべて数値情報として入力されたときに，その対象がディスプレイに表示されるわけです．ただし当時は 2 次元の線画のモノクロ表示からのスタートでした．その後コンピュータ性能の驚異的な進歩に合わせるように，1970 年代から 1980 年代にかけて，光と影のビジュアルシミュレーションとしての CG の進化がありました．1980 年代には 1600 万色のカラーディスプレイが登場し，1980 年代後半には線光源や面光源なども考慮した，ぼやっとした半影の表現なども可能になりました．また人工物だけでなく，山や雲などの自然物もリアルな表現が可能となりました．ただし 1 枚の画像生成に当時の大型コンピュータで数時間を要する場合もありました．当時の CG としては，1985 年に登場した『スーパーマリオブラザーズ』(任天堂)など，いわゆるドット絵の 2 次元デジタルゲームが一般に知られていました．まだ CG という言葉は専門用語であったかもしれません．

1990年代に入ると，実在する建物などをリアルに表示するには，シミュレーション的手法だけでなく，コンピュータビジョン(Computer Vision，CV)の手法によって2次元の複数の画像情報を用いて，形，色，テクスチャ(表面の質感)を推定するイメージベースレンダリングと呼ばれる方法論が出てきました．これにより入力する形状モデル作成の労力が大幅に軽減されます．1999年の映画『マトリックス』(ワーナー・ブラザーズ)の一部のシーンは，この方法を用いて作られました．CGとCVの融合と発展は，この頃から加速しました．1990年代終わり頃からGPU(Graphics Processing Unit)が開発され，CG特有の計算処理がチップ化され高速化されました．これにより高品質リアルタイムCGの実現という大きな研究テーマへとつながりました．一般の人々にとって，見て楽しむCGから使えるCGとなるためには，リアルタイム性は不可欠です．

2000年以降のCGは表示対象を広げつつ，エンターテイメントを越え，医療・広告・教育など，さまざまな分野に浸透してきました．また技術的にはCVに分類されるかもしれませんが，2次元の画像や線画情報から3次元情報を推定する手法がいろいろと開発され，AIやディープラーニングとの繋がりも出てきました．コンピュータのハードウェア・ソフトウェアの進化，CVやAIなどの最先端技術との協働により，CGは発展し続けています．そして何より数学が，それを支えているのです．

本書はCG技術の基礎から始めて，最先端のCG研究事例までを紹介します．その際に重要な役割を担う数学の紹介とCG研究での役割も述べます．もちろんCGも数学もそれぞれ扱う対象は実に多岐にわたります(数学は歴史も古く，CGとは比べものになりません!)．ここでは各章を簡単に紹介しながら，本書で取り上げるCG技術とそれを支える数学的アプローチを見てみましょう．

第1章 CGやアニメのための簡単な数理モデル　レンダリングやアニメーションの基本的考え方を中心に，CGや映像表現の基礎についてまとめます．

第2章 トゥーンシェーディングの数理モデル　3次元CGで構築した世界をアニメ風にレンダリングする基本手法と，実用上の問題を紹介します．また記号的に用いるハイライト制御が困難となる問題がありますが，これに関する解決策としての数理モデルを紹介します．

第3章 フォトリアリスティックレンダリング　リアルなCG画像生成の基礎となるレンダリング方程式とその基本的な解法を概説します．その応用として，周囲光を画像データとして与えた場合のイメージベースライティングと呼ばれる手法も紹介します．また近年急速な進展を遂げているディープラーニングの応用として，複数視点からの画像データをもとに，新たな視点からの画像生成を実現する手法も取り上げます．

第4章 1枚の画像から3次元の世界へ　1枚の画像データを入力として，それから想像できる簡単な3次元シーンモデルを構築するためのユーザーインターフェース(User

Interface，UI)について紹介します．数学的に高度なものは何も出てきませんが，言わば人間の想像力を引き出すインターフェースの例として紹介します．その後の CG 研究にも影響を与えている技術です．

第 5 章 カメラと 4 元数とファイバーバンドル　仮想の 3 次元シーンを含む空間にカメラをセットし，そこから見える映像が CG ですが，この章ではこのカメラ制御がテーマです．カメラの回転を含む動きの制御に 4 元数やファイバーバンドル(ファイバー束)といった数学的概念が活躍するところを紹介します．

第 6 章 形やテクスチャの特徴づけ　自然物や 3 次元計測技術を用いて得られた複雑な形状の特徴づけに関する 2,3 の手法を紹介します．確率過程やエントロピーといった確率論・情報理論的な概念が，形の特徴づけに使われるところに数学的発想の面白さがあると思います．

第 7 章 物理ベースのアニメーション　アルマジロ(のような形のキャラクター)が飛び回ったり，空に拡がる雲の形を表現したりというところにも，リアリティを追求すれば微分方程式が使われます．作り手の意図と，自然法則に基づくリアリティの間でうまくバランスを取るアイデアを紹介します．

第 8 章 キャラクターアニメーション　手描きのキャラクターの作り方から始めて，人間やキャラクター，あるいはその群集を 3 次元 CG で表すための基本手法を紹介します．この中には 1990 年代の AI 技術を用いたアニメーション手法も含まれます．後半はキャラクターの表面形状が骨格構造の変化に対応してどのように変形させるか，についての数理モデルを紹介します．特に動径基底関数(Radial Basis Function，RBF)と呼ばれる一連の関数たちと，その数学的背景に触れます．

第 9 章 フェイシャルアニメーション　人間の顔の細やかな表情や動作を CG で表すことは大変難しい挑戦的な課題です．本章では，この課題に対する数学的な取組みや，例えば実際の人間を計測してモデル化する技術や背景にある心理学的なアプローチも紹介します．後半では，フェイシャルアニメーションのスタンダードな手法の一つであるブレンドシェープを取り上げます．フェイシャルアニメーションに挑むアーティストセンスとブレンドシェープ技術のコラボレーションを具体的に紹介します．

これらの章のうちいくつかは，『数学セミナー』(日本評論社)連載「CG につながる数学」(2009 年 4 月号〜9 月号)，「続・CG につながる数学」(2010 年 10 月号〜2011 年 3 月号)で取り上げた話題の中から，今後も基本的で重要な項目を選びました．各章の後半は単行本化に際して話題を追加し，記述を詳しくかつ読みやすくしました．第 5 章は『数学セミナー』2013 年 1 月号「単位 4 元数空間 ── コンピュータグラフィックスにおける被覆の応用」(特集「被覆のはなし」)から取っています．特に，第 1 章後半，第 3 章後半，第 6 章前半，第 8 章後半，第 9 章は今回書き下ろしました．また，やや専門的な数学用語については，必要となる

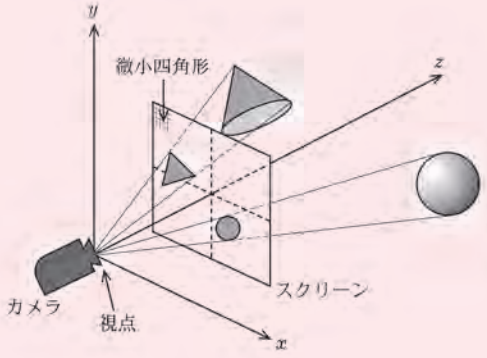
章末のコラムに解説してあります．したがって本書は，上記『数学セミナー』の掲載内容をベースとしつつも，最新の話題を含んだ，CG と数学を語ったユニークな内容となっています．なお加筆あるいは書き下ろし部分は，陰に陽にリアルタイム CG 技術の発展の影響が大きいです．ただし本書ではリアルタイム CG のための GPU プログラミングについては割愛しています．むしろそのリアルタイム化を実現する上で脚光を浴びている数学的手法を取り上げました．

本書で取り上げた CG 技術のほとんどは，アニメーションへの適用例をみることでその真価がわかります．さいわいなことに今日では，それらのアニメーションは，インターネットを通じてみることができます．本書ではその URL も本文中に示してありますので，ぜひアニメーションをご覧いただきながら，読み進めてください．これは 10 年前ではまったくかなわなかったことです．筆者としては，大変良いタイミングで本書を出版することができたと考えています．

なお本書では，このような CG の問題にはこのような数学的手法が使われている，といった辞書的・網羅的な解説というより，表示対象を具体的に絞り込んだ上での技術的な課題と解決策，およびその数理的背景を述べています．また，例えば第 4 章のように，数学的には整理されていない，問題提起のような内容も含まれています．数学を専門とする方々，より深く数学的側面を知りたい方には，多くの文献を取り上げてありますので，それらを参考にしていただければ幸いです．では本論にはいりましょう．

2023 年 6 月

安生健一



第 1 章

CGやアニメのための簡単な数理モデル

この章では，そもそもCG映像とは何であり，どうやって作成するのか，について概説します．物体表面の質感を表す簡易モデルや，アニメーションを作る上で最も基本的なキーフレーム法についても紹介します．

まずいくつかの基本的なCGの用語や概念を説明します．本書ではCG研究の最先端まで紹介したいので，厳密な定義は用いずに，なるべく最短距離に近いコースを通って，最新の話題に到達できるよう努めてみます．

CGとは，その名の通り，コンピュータを使って生成された画像，あるいは画像生成技術のことです．この場合の画像とはデジタル画像のことで，コンピュータ内で2次元配列として記憶されます．配列の各要素は画素，ないしはピクセルと呼ばれ，色情報が蓄えられています．各画素は画像の番地 (i, j) をインデックスとして，その番地での色情報が格納されます．カラー画像であると，例えば赤，緑，青の色(の強さ)がそれぞれ数値で表されます．各3原色を1バイトで，合計3バイトで色階調を表せば，数値的には約1600万色が表せます．$(255, 0, 0)$ は赤，$(255, 255, 0)$ は黄色，$(128, 128, 128)$ は灰色，といった感じです．

ところで，現実の3次元世界はデジタルカメラなどを通して，ペンや絵筆で描いた風景やアニメの原画などの2次元世界はスキャナーで読み込んで，デジタル(画像)化できます．本書では，CGによって2次元ないし3次元の世界を表現するのですが，その表示対象はこの画像化のプロセスを経て扱うのではありません．本書における表示対象は，2次元空間または3次元空間内の仮想オブジェクトとして，すべてコンピュータ内に数値データとして定義されています．例えば，図1.1のように仮想的な3次元世界に球があれば，ある座標系のもとで球の中心位置と半径が指定されます．表面の色も，この「球」の属性として指定します．同図手前にある平面(四角形)は投影されるスクリーンと考えてください．3次元CGのカメラモデルでは，たいてい「透視投影」を行います．すなわち，カメラから物体へ視線を伸ばし，その視線とスクリーンの交点に物体を描画する投影方法を用います．表示される3次元シーンをスクリーン上に映し出すには，光源情報(種類，強さ，位置)と，カメラ情報(位置，画角など)が必要です．カメラ情報は観察者の視点情報と考えても良いです．画角(視野角)はカメラが望遠か広角かを指定するもので，図1.2にあるように，視点からスクリーンをカ

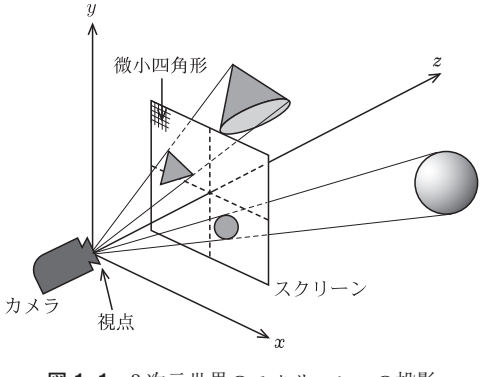


図1.1 3次元世界のスクリーンへの投影

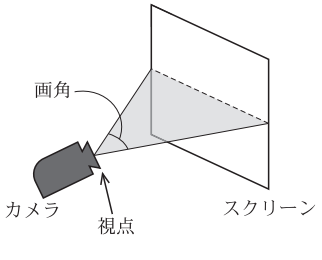


図1.2 画角

バーする角度のことです．同図ではスクリーンの水平方向を考慮した水平画角を示しています．同様に垂直画角，対角画角も定義できますが，CG ツールによって画角の定義は異なります[128]．映像制作現場では，表示物体は球一つ，ということはまずありませんから，例えば巨大な都市を表示しようとしたら，ときに億単位以上の物体(部品)からなるシーンを扱うこともあります．さらに時間経過とともにこの入力データが変化すれば，出力は画像の列となりますが，これを連続して表示したものがアニメーション(animation)です．映画ですと 1 秒間に 24 フレームの画像が表示されます．デジタルゲームとなると，1 秒間に 60 フレームかそれ以上です*1．

1.1 レンダリングと隠面処理

レンダリング(rendering)とは，上述のようなシーンデータからスクリーンとして定められたデジタル画像の各画素の色情報を得るまでの処理全体のことです．レンダリング処理の概要は次の通りですが，後の章にてより詳しく説明します．ここでは簡単のために，3 次元の表示物体を想定し，それらは微小な多角形の集合であるとしましょう*2．図 1.1 に示されるように，スクリーンを微小な四角形を単位とする集合と考えます．全体としては通常大きな長方形になります．スクリーン上の微小四角形は先述の画素に対応し，この微小四角形に届く 3 次元シーン内の物体の色情報が蓄えられます．通常は透視投影を行い，**隠面消去**と呼ばれる処理で，各画素に届く，すなわちカメラに一番近い物体を決定し，その物体の色情報が決められます．基本的な隠面消去法のひとつに**デプスバッファ法**，あるいは **z バッファ法**と呼ばれるものがあります．各画素における各表示物体との距離を深さ(depth)と呼びます*3が，まずこの値を各画素に蓄えるための 2 次元配列を用意して，これを z バッファと呼びます．新しく物体が追加されると，この z バッファが更新されます．もしこの新しい物体の深さが，もともと蓄えられている深さより小さければ，それだけ画面に近い，ということですから，この新しい値が蓄えられ，対応する色情報も更新されます．シーンに登場するすべての物体を考慮して更新された z バッファの情報に基づきスクリーン画像の各画素の色が

＊1　フレーム(frame)とは，連続して作成・表示される画像の枚数のことを指します．1 秒間に何枚の画像が表示されるかをフレームレートと呼びます．例えば映画ですとそのフレームレートは 24 fps (frame per second)です．

＊2　もちろん，雲や煙のように密度分布として捉えた方が良い表示対象もたくさんあります．これらの表示対象にはボリュームレンダリング(volume rendering)という手法が適用されます．この手法については 3.6.2 節や 7.3 節をご覧ください．

＊3　CG で用いられる座標系は，図 1.1 に示されるように，スクリーンに対して垂直に向かった深さ方向を z 軸とすることが多く，これにちなんでこの手法は z バッファ法と呼ばれています．

決定されます．表示物体が半透明ならどうするか，影はどう計算するのか等々いろいろと疑問に思われる点もあるかと思いますが，それは必要に応じて以下の章で説明していきます．

1.2 局所シェーディングモデル

隠面処理ができたとして，物体表面の色はどう決めたら良いでしょうか？ 光源情報やカメラ情報は与えられますが，さらに物体それぞれに固有の色があります．ハイライト(最も明るく見える部分)などは時々刻々とカメラの位置や光源の位置などとともに変化します．**シェーディング**とは，物体の任意の点における色を決定することを意味し，シェーディングモデルはそれを実現する簡易物理モデルのことです．ここではもっとも基本的で，市販のCGツールにて用いられている**局所シェーディングモデル**を紹介します．ここでいう局所とは，光の反射屈折などを1回だけ考慮する，という意味です．より複雑な陰影をリアルに表現するためには大域的な照明モデルが必要となります．これについては第3章で紹介します．

物体上の点Pにおける輝度をIとすると，厳密なことはさておき，Iは先の3原色のそれぞれに対応する成分を持つ3次元ベクトルであり，その点での色そのものであると考えられます．Iは，環境光成分I_a，拡散反射光成分I_d，鏡面反射光成分I_sの3つに分けられ*4，その和として表せます．本書の範囲ではIの取り扱いは各成分でまったく同じなので，以下ではスカラー値をとるものとします：

$$I = I_a + I_d + I_s. \tag{1.1}$$

この式を少し詳しく見てみましょう．光源から出た光は，物体間で反射を繰り返し，その光がまたさらに物体を照らしてゆくので，直接光があたらない部分でも見ることは可能です．このような光の効果を非常に粗く近似したものが**環境光**であり，それによる輝度が上記のI_aです．環境光は**周囲光**とも呼ばれ，周囲から来る一様な光のことです．したがってI_aは，光源の向きや強さ，カメラの位置には依存しません．次に**拡散反射**ですが，これは石青やチョークの表面での反射のように，どの方向から見ても輝度が一定となるような反射のことです．例えば平行光線があたっている場合には，**Lambert(ランバート)の法則**により，物体表面の点Pにおける物体の法線$\boldsymbol{N}$と光源の向き$\boldsymbol{L}$との内積値$\langle \boldsymbol{N}, \boldsymbol{L} \rangle$に比例して求まります．ここで$\boldsymbol{N}$も$\boldsymbol{L}$も大きさは1に正規化した単位ベクトルとします．また$\boldsymbol{L}$の向きは図1.3(次ページ)のように，物体から離れる方向です：

$$I_d = k_d I_i \langle \boldsymbol{N}, \boldsymbol{L} \rangle = k_d I_i \cos\alpha. \tag{1.2}$$

ここでk_dは拡散反射率で物体に依存して決まる係数であり，I_iは入射光(平行光線)の強さです．

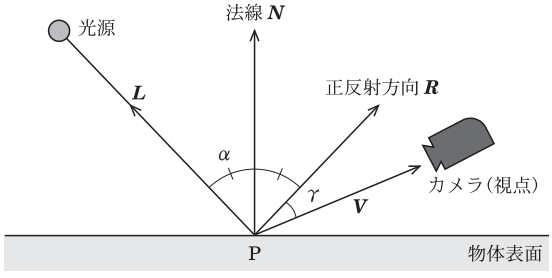


図 1.3　局所シェーディングモデル

図 1.4　Phong のモデルを用いたハイライト表現：(a)ハイライトなし，(b)シャープネス小(n が小)，(c)シャープネス大(n が大)．

最後に**鏡面反射**です．これは物体表面からの直接反射のことで，いわゆる**ハイライト**と呼ばれるものです．点を求めるのによく用いられるのは次の **Phong(フォン)のハイライトモデル**です：

$$I_s = k_s I_i \langle \boldsymbol{R}, \boldsymbol{V} \rangle^n = k_s I_i \cos^n \gamma. \tag{1.3}$$

ここで $\boldsymbol{R}$ は，光源の正反射方向を示す単位ベクトルで，k_s は鏡面反射率，$\boldsymbol{V}$ は視線方向を表す単位ベクトルです．n はハイライトのシャープさを制御するパラメータで，この値が大きいほどシャープなハイライトが出ます．図 1.4 に簡単な適用例を示しました．実は Phong のハイライトモデルは物理的に厳密なモデルではなく経験則に基づいたものです．したがって k_s や n の調整には試行錯誤が必要ですが，結果がもっともらしいので現在もよく使われています．

＊4　I_a の添字 a は ambient light (環境光)，I_d の d は diffuse light (拡散反射光)，I_s の s は specular light (鏡面反射光)にちなみます．

1.3 テクスチャとマッピング

前節までに，3次元シーンがどのようにレンダリングされるか，物体の色情報を決めるシェーディングモデルとは何かを，ごく簡単に紹介しました．まだアニメーションの話には触れていませんが，本書の主題であるCGと数学の接点について簡単な例を紹介したいと思います．

1.3.1 さまざまなマッピング

前節で述べたシェーディングモデルはいわば物理的な考察に基づくものであり，特に数学との直接的なつながりは感じられないかもしれません．映画やデジタルゲームなどの映像制作でのCGは，効率と表現力の向上という，ある意味で相反する目的を持っています．ゲームはいうに及ばず，CGを用いていかに効率よく目的にかなった映像を作れるかが重要です．一方SF映画では現実には存在し得ない世界やクリーチャーを「リアルに」描くとか，『ポケットモンスター』のようなアニメの場合は，手描き表現では大変困難か，あるいは不可能な映像表現を作り出すのに用いられます．球の表面は，各点で法線が解析的に求まりますから，前章で述べたシェーディングモデルで陰影づけをし，レンダリングすることは容易です．しかし現実世界にあるものはもっと複雑です．例えばリンゴをリアルに表現するには，その表面の質感も表現する必要があります．これを簡単に実現する手法として**テクスチャマッピング**(texture mapping)がよく使われます．この方法は，**テクスチャ**と呼ばれる2次元画像により物体表面の色や質感をあらかじめ作成し，これを表示対象の3次元モデルに写像するというものです．ここでいう3次元モデルは，曲面や多面体(CGの世界ではしばしばこれを「ポリゴン(polygon)」と英語表記そのままで呼びます)でできています．リンゴとかプチトマトとか，表面がツルツルのものでしたら，テクスチャマッピングを用いれば，それらしい3次元CGの果物が作れそうなことは想像できると思います．ただし，つやつや度を制御するために，前述のハイライトモデルの指数nを調整する必要があります．ではオレンジの表面のように非常に細かな凹凸を持つ物体の場合はどうでしょうか？ 実際のオレンジがあれば，その表面の凹凸を3次元計測して多数の頂点の集まりとして表し，それから例えば微小ポリゴンの集まりとしてオレンジの表面モデルを作ることができます．しかしこれでは手間がかかり，ちょっと大げさです．拡散反射の式(1.2)を眺めてみると，物体表面上の点の法線と光源との内積を求めて輝度を計算していますから，要するにこの法線ベクトルをうまく定義してあげれば，オレンジなど複雑で微小な凹凸を持つものが表現できそうです．これを実現する方法は**バンプマッピング**(bump mapping)と呼ばれています．図はその原理を簡単のために1次元で示しています．図1.5(a)(次ページ)が，もともとの法線ベクトルの分布であるとします．ある物体の表面をミクロスケールで見ると，この図(a)のように近くの法線は

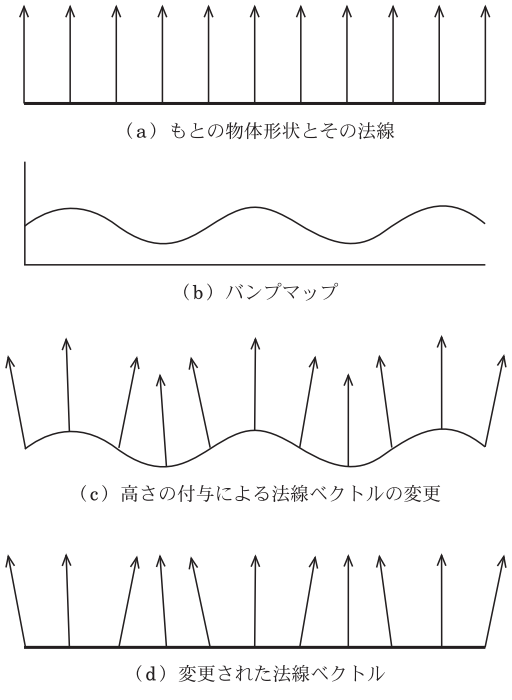


図 1.5　バンプマッピングの原理

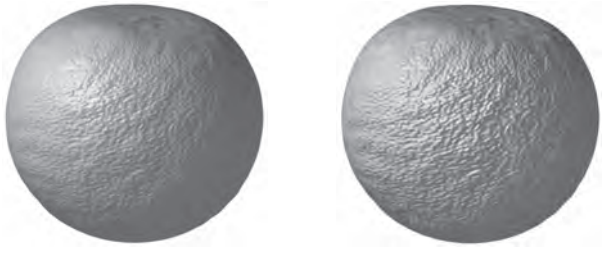


図 1.6　バンプマッピングの適用例：左右の画像は，同じ幾何モデルに異なる法線ベクトルの分布を与えて生成されている．

ほぼ平行であると考えましょう．さて表示物体の表面の凹凸を表すための画像を用意します．これは**バンプマップ**と呼ばれます(図 1.5(b))．次にコンピュータが，バンプマップの輝度値に従って物体表面に擬似的な凹凸を生成して，その法線ベクトルを計算します(図 1.5(c))．レンダリングの際には，この法線情報をもとにして最終的な輝度値を求めます(図 1.5(d))．バンプマッピングでは法線情報のみを変更し，表示物体の幾何情報(もとの曲面，ないしはポリゴンモデル)は変更しません．図 1.6 は，図 1.4 に用いた形状モデルの上にバンプマッピ

ングを施した例です．一見表面は凹凸に見えていますが，そのシルエットを見るともとの形状のままになっています．マッピングにはこれら以外にもたくさんの手法があります．例えば，表面の幾何情報そのものを変更する**ディスプレイスメントマッピング**(displacement mapping)という手法もあります(例えば[128]を参照してください)．

1.3.2●見た目をモデル化する

こうしたさまざまなマッピング技術は，簡単便利にCGモデルが作れる，という意味で非常に大切です．効率よく表現力豊かな形や質感を実現できます．しかし物理法則には基づいていません．写真などの画像をマッピングして表面の質感表現をしたり，法線をわざと変更して擬似的に凹凸を変更したり，これらは一体何に基づいた方法なのでしょうか？ 大雑把にいってしまうと，これらは人間の見たままの感覚を数値情報，すなわちデジタル化した結果と考えられるのではないでしょうか．CGの研究開発とは，人間がさまざまなものを見たときの感じ方，感覚をうまくモデル化することでCG映像にリアリティや表現力を付与しようする試みのように思えます．

このことはCGによるアニメーションについてもいえます．ただし流体の動きなど，自然現象に関する見た目のモデル化には，やはり物理学の助けが必要です．例えばNavier-Stokes(ナヴィエ-ストークス)の方程式を簡易化して解いたりします．とはいえその簡易化は，単に効率化のためだけではなく，人間が流体を見て感じるリアリティのエッセンスを獲得するための過程であろうと思われます．

1.4 キーフレームアニメーション

本書の後半は3次元コンピュータアニメーション(3D computer animation)が主たるテーマとなります．そこで本節では，アニメーションの基本であるキーフレームアニメーションについて概説します．この最も一般的なアニメーション手法の起源はパラパラ漫画にあります．手描きアニメの基本ともいえるでしょう．

これを作るには一枚一枚の漫画のコマ(CGの言葉でいうとフレーム)を描く必要があります．テレビや映画のアニメでは実写ほどのフレームレートは望まれませんが，1秒間に12フレームとか8フレームなどがよく使われます*5．細かい微妙な動作を短い時間で表すには，当然1秒あたりのコマ数は多く必要になります．そうしないと動きが飛んでしまって，カクカクしたアニメ映像になってしまうからです．さてこのパラパラ漫画の原理で90分のアニメ映画を作ることを考えましょう．1秒間につき8コマだとしても，相当なコマ数の静止画を描かなければなりません．どのような制作工程になるかというと，まず前段階として，脚本が完成し，それをもとに絵コンテが作られます．絵コンテ(storyboard)は，実写でもアニ