Scheme による 3D 図形の構成的制作

古川 孝太郎　糸山 克寿　吉井 和佳　奥乃 博

本稿では，LISP 系言語の 1 つである Scheme を用いた 3D 図形言語の設計と実装について述べる．効率的にプログラミングを行う上で，手続きやデータ構造の抽象化の概念を正確に理解しておくことは重要である．プログラミングの代表的な教科書 SICP (Structure and Interpretation of Computer Programs) においては，プログラムの階層構造を直観的かつ視覚的に捉えることを目的として，Scheme を用いた平面図形構成のためのシステムである図形言語が利用されてきた．我々は Scheme 処理系として JAKLD を用い，オリジナルの図形言語に立体図形構成のための拡張を加えた．さらにプログラムに記述された立体フラクタル図形を 3D プリンタを利用し実体化することで，プログラミング初学者の大学一年生が手続きとデータ構造を効果的に習得するための教材として利用した．

We have developed a picture language system for constructing 3D models in Scheme, one version of Lisp. Understanding both procedural abstractions and data abstractions is important for all people who learn to write efficient and compact source codes. A picture language has been the most effective way to understand those concepts, because it can present structures of programs as intuitively and visually understandable 2D figures, which was originally introduced in a legendary book of computer science, SICP (Structure and Interpretation of Computer Programs). We designed and implemented the extended picture language for 3D models using JAKLD and enabled the system to export models in 3D printable format, in order to deepen learners' understanding of abstractions by analogy of spatial fractal structures of substantial models. We report the use of our system as a teaching material in a lecture for first-year undergraduate students.

1 序論

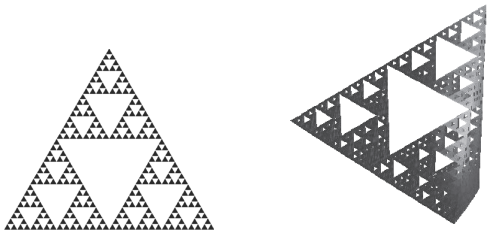
本研究は，京都大学工学部情報学科の開講科目「アルゴリズムとデータ構造入門」[23] の講義を改善し，3D 図形，特に図 1 のような立体フラクタル図形の構成と造型を通して，手続き・データ抽象化への理解を促進するシステムを開発する．プログラミング言語 Scheme [2] 上で 2D 図形を構成的，再帰的に描画可能なシステムである図形言語 [3] を取り扱い，図 2 に示す 3D 図形の構成と描画，3D プリンタによる造型のための入出力という機能を拡張する．実現したシステムを講義で取り入れ，学生から随意課題として提出を受けた作品を造型したモデルによりフィードバックを行う．本研究の成果物として，プログラムのソースコードを Web 上に公開する．

プログラミングにおける抽象化の概念は，人間にとって理解しやすく再利用性の高いソースコードを作成する上で極めて重要である．抽象化には手続き抽象化とデータ抽象化の二種類があり，前者は計算機による一連の計算プロセスに抽象的な名前をつけ，またパラメータを引数として明確化し手続きとして定義することで，それがユーザにとってどのような意味を持つ処理であるのかを明示する行為である．ゆえに手続き抽象化は，目的に基づいてプログラムを作成する上でそれを実現するコードをどう記述すればよいか判断し，あるいは既に作成されたプログラム中の記述を理解する助けとなる．また呼出しという

Picture Language for Constructive 3D Modeling in Scheme.

Koutarou Furukawa, Katsutoshi Itoyama, Kazuyoshi Yoshii, Hiroshi G. Okuno, 京都大学大学院情報学研究科, Graduate School of Informatics, Kyoto University.

コンピュータソフトウェア, Vol.32, No.4 (2015), pp.31–49.
[ソフトウェア論文] 2014 年 10 月 29 日受付.



(a) Sierpiński の三角形 [30]　(b) Sierpiński の四面体 [20]

図 1　図形言語の 3D 拡張

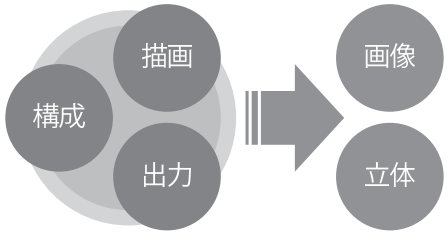


図 2　本システムの全体像

形でその記述を参照して利用することができるから，誤りが含まれる場合にそれを修正するコストを減らすことにもつながる．データ抽象化はデータの実装と意味との分離であり，インタフェースとしての構成子や選択子を用意することで，実装上どのようなデータ構造に基づいているかを意識することなく，そのデータを利用できるようにする行為である．これも手続き抽象化同様にユーザにとって把握すべき情報を減らすという意味で，プログラムとソースコードへの理解を容易にする．例えば線分を描画する手続きを利用し，その組合せによって正方形を描画する手続きを定義することは手続き抽象化の 1 つである．また正方形というデータ構造を，中心座標と一辺の長さを値に持つリストとして定義し，それらを引数に取る構成子と，それぞれをリストから取得する選択子を定義することはデータ抽象化の 1 つである．

「アルゴリズムとデータ構造入門」は，主に情報学科に所属する一年生を対象とした，プログラミングの基礎を学ぶことのできる講義である [34]．教科書に SICP (Structure and Interpretation of Computer Programming) [3] を用い，Scheme プログラミングの実践を通して学習を進める．図形言語はこの講義に含まれるテーマの 1 つで，プログラミングの本質的内容である手続き抽象化とデータ抽象化を学ぶために最適な題材の 1 つである．これは，例えば「線分を描画する」や「図形を分割する」といったプリミティブな手続きの組合せによって複雑な図形を構成できるため，手続きの呼出しを視覚的に捉え，再帰呼出しや階層化の概念を直観的に学べるからである [32]．

Scheme は Lisp [21] 方言の 1 つであり，コンパクトな仕様と高い表現能力を兼ね備えたプログラミング言語である．高水準言語に属し，また対話的実行環境を備えトライアルアンドエラーの開発スタイルを容易とするために，プログラミング初学者にも学びやすく扱いやすい．そのため上記教科書の他にも優れた解説書 [6][8][31] や処理系 [9][10][29] が数多く存在し，教育の現場でも多数取り入れられている [27][28]．また一般に，プログラミング学習の過程で Scheme を最初の言語として学ぶことは，筋のいいプログラミング技能を身につけ，計算機科学への興味を深めるために有益であると言われている [4]．我々は，SICP の内容に準拠し拡張性に富んだ処理系である JAKLD [33] を使用し，図形言語の設計理念に沿って教育と学習上の利点を生かしつつ，3D 図形の構成的な作成と立体造型可能な形式への変換機能を実現する．

本研究は，図形言語の 3D 拡張とともに，その上で作成した図形を 3D プリンタによって造型するためのインタフェースを提供する．3D プリンタは昨今大いに注目を集めるデバイスであり，可塑性樹脂を熱で溶かし平面形状を積層しながら立体形状を造型するものが主流である．これはラピッドプロトタイピングのための主要なツールとして広く利用されている [7][16]．従来の講義では平面図形の描画しかできなかったが，作成した 3D 図形を実際に手に取ることのできるモデルとして実体化し学生へフィードバックを与えることで，そのプログラムの構造への理解を深めるとともに講義内容の積極的な学習を促進する．

本稿は次の構成を取る．まず 2 節でオリジナルの図形言語や図形の記号的処理に関連する先行研究について概観する．次に 3 節で 3D 図形言語の要求と設計について述べ，4 節においてその実装を構成，描画，

入出力の各機能に分けて詳説する．さらに 5 節で本システムの利用実績とユーザによる反応を挙げて効果及び改善点を考察し，6 節で結論を述べる．付録に本稿で紹介した 3D 図形作例のソースコードを載せる．

2 関連研究

本節では関連する研究を紹介し，その課題を明らかにする．以下ではまずオリジナルの図形言語を概説し，3D 図形を構成的にモデリングする CSG (Constructive Solid Geometry) の手法，図形を計算機上で取り扱うため記号処理的手法を開発した Lisp 系言語における初期のアプローチに触れる．図形言語は CSG や記号処理を指向する流れの中に直接に位置づけられるものではないが，これらの目指す理念や特長は多く共通する部分があるため，我々のシステムを設計する上で大いに参考となる．

2.1 図形言語

SICP によって導入されるオリジナルの図形言語は，図形を描画し変換するための一連の手続き群として定義される，Scheme の言語拡張である．図形言語はペインタとフレームという 2 つの概念によって特徴づけられる．ペインタとは図形そのものの情報，すなわち線分であれば始点と終点，多角形であれば頂点の座標といった情報を保持し，評価することでその図形を描画する手続きである．この評価時に引数として取られるのがフレームであり，それはペインタの持つ図形についての変換を意味する．フレームは具体的には，その変換により原点と 2 つの基底ベクトルの移される先の座標によって表され，これらをそれぞれ $\mathbf{o}', \mathbf{u}', \mathbf{v}' \in \mathbb{R}^2$ と表すとすると，フレームは式 (1) に表される二次元アフィン変換行列 [14] $\mathbf{A}$ に相当する．

$$\mathbf{A} = \begin{bmatrix} u'_0 - o'_0 & v'_0 - o'_0 & o'_0 \\ u'_1 - o'_1 & v'_1 - o'_1 & o'_1 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

ペインタの評価時にはそのペインタの持つ図形の各点 (x, y) に対して，式 (2) が適用され，新たな図形

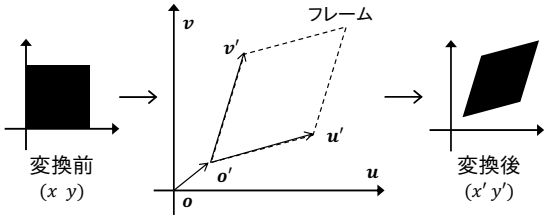


図 3 フレームのペインタへの適用

中の点 (x', y') が得られる．

$$(x', y', 1)^{\mathrm{T}} = \mathbf{A}(x, y, 1)^{\mathrm{T}} \tag{2}$$

ペインタの変換は図 3 に表される．

ペインタの概念は上記のとおりであるが，実際にそれを構成する手続きは実装に依存する．本システムで使用する処理系 JAKLD は `segments->painter` と `vertexes->painter` という 2 つの手続きを持つ．`segments->painter` は線分のリストを受け取ってその全てを描画するペインタを構成する手続きであり，`vertexes->painter` は頂点の座標のリストを受け取って多角形を描画するペインタを構成する手続きである．一方フレームは手続き `make-frame` によって定義され，これは変換を表現するために原点と基底ベクトルの変換先の座標を引数に取る．ペインタの合成は，複数のペインタを順に評価することで図形を塗り重ねることによる．

図形は一辺を 1 とする正方形状の領域を表すウィンドウに描画される．例えば単一の多角形を描画したければ，この領域に含まれる頂点のリストを `vertexes->painter` へ与えて構成したペインタに，恒等変換を表すフレームを与えて評価する．フレームは多角形を変形させながら複数描画したいというような場合に有益であり，定義された 1 つのペインタに対して異なるフレームを与えることで図形の定義の重複を防いだり，再帰的に変換を施しながら描画することが可能である．

ペインタにフレームを適用する際，即座に描画するのではなくその変換後の図形を保持するペインタを構成することも可能である．これは新たにフレームを引数に取り，先に適用したいフレームと合成した上でそれを元のペインタに適用する手続きとして定義す

ることによる．フレームの合成は単に変換の合成として，それらを $\mathbf{A}_0$ と $\mathbf{A}_1$ と表すとすれば $\mathbf{A}_1\mathbf{A}_0$ として得ることができるから，同様にフレームの合成を繰り返すことで，図形に変換を繰り返し適用した複雑なペインタを構成することが可能となる．

図形言語がプログラムの構造化を学ぶ上で有益であるのは，コード中の記述と図形の部分とが対応づけやすく，構造を視覚的に捉えられる点と，単純な図形から再帰構造を持つフラクタルのような複雑な図形まで能力に合わせて描くことができ，段階的に学習できる点である．前者は図形それ自体を表すペインタと，それに対して適用される変換であるフレームという役割分担が明確であることによっている．後者はペインタ定義のために提供される手続きが一般的で単純なものであり，その組合せによって複雑なペインタを構成するという思想に基づいている．複雑な図形を描く上で手続きとデータの抽象化はコードの可読性や再利用性を高める上で重要となるから，試行錯誤しながら図形を定義することは抽象化の学習に役に立つ．

2.2 CSG の概要とその実現

CSG (Constructive Solid Geometry) [25] は 3D 図形を構成的に作成するアプローチである．これはプリミティブな立体図形に対して，演算を組み合わせて所望の形状を得るというものである．例えば 2 つの図形に対する加算は，それらの図形の境界及び内部に含まれる点の和集合からなる図形を返し，ほかに差集合を表す減算，積集合や排他的論理和等が提供される．また図形の拡大縮小や平行移動，回転といった操作も提供され，これらの組合せにより理論的には任意の図形が構成可能となる．

CSG を実現する多くの実装がライブラリや CAD として公開されているが，そのうちの 1 つが OpenSCAD [17] である．OpenSCAD はオープンソースの CSG CAD であり，簡易なスクリプトによって図形の記述が可能であり，また 3D モデルデータのデファクトスタンダードな形式である STL (Standard Triangulated Language) [1] 形式への変換機能も備えている．3D モデリングのためのドメイン固有言語である OpenSCAD スクリプトは高階関数がない，末尾再帰の最適化がなされていない等の機能的な非力さがあるが，本研究では Scheme の言語拡張である図形言語との対応付けによりこれを補うことができ，CSG CAD としての機能を活用しながらプログラミング教育の用にも供し得ることを示す．

2.3 図形情報の記号処理

Lisp 系言語においてはその記号処理に長じている性質を生かし，文字や言語の情報を扱うのと同様にして図形を記号処理的に扱う取り組みが早くからなされてきた．これは図形の要素の形状や組合せ的な構造を明確に表現する記法を定義し，その上で図形の分解，要素の抽出，再構成等の操作を加えて図形の本質的な情報の処理を可能とするものである．

PAM (PAttern Manipulating) Graphics System [19] は文字情報と図形情報をともに計算処理の俎上に載せる試みであり，MacLisp [22] 上に実装された．このシステムは人間同士のコミュニケーションで見られる文字と図の両方による情報のやりとりを計算機によって取り扱えるようにすることを企図している．Visual Grammar Notation [18] はこうした文字と図形からなる情報を統一的に表現する文法を実現するものであり，その後，電子回路図や組織図，数式といった図形的な要素によって表される人工的に作られた言語の構造の解析のための研究 [5] を導いている．

記号処理的アプローチは，応用上の目的に特化する記法を言語のみならず図形も用いて設計するか，その図形による構造化された表現を読み取り解析する方法を開発することに主眼がおかれているものである．よってこれらの言語はプログラミング教育に直接生かせるものではないが，図形を個々の要素と構造に分解し解析しようという着眼点は，図形とプログラムの類推からよりよい抽象化を学ぶための言語を設計する本研究においても重要である．

ASAS (Actor/Scriptor Animation System) [26] は 3D 図形を構成し，その動作を記述することのできる言語を開発することで，構成的に 3D アニメーションを作成することを可能とするシステムである．簡易な記法で 3D 図形の移動や回転，画面のズーム等の映像上の効果を得る処理を記述できるという利点を持ち，

個々の動作を行うモジュールが高い独立性を持っているため，複雑な映像を表すプログラムの構成を容易にしている．ASAS は現在では用いられていないが，商業上の利用もなされた高い完成度を持つシステムであり，その言語設計において提供される手続きの選択や，抽象化方法は示唆に富んでいる．

LOGO [11] はプログラミング教育における利用を目的としてプログラムの視覚的な理解を実現するため，Lisp 上で実装されたシステムである．LOGO は図形の描画機能を充実させており，中でもタートルと呼ばれるカーソルを用いた仕組みを導入している点が特徴的である．タートルはユーザによってプログラムされた動作を取り，その軌跡が図形として描画される．オリジナルの LOGO を 3D に拡張して図形の 3D プリンタによる造型を可能としたシステム [15] も提案されている．記述した内容がアニメーションによって逐次，視覚的にフィードバックされるため，特に学習の初期におけるプログラミングという概念の導入を支援するのに有効なシステムである．本研究では LOGO の利用が得意とするプログラミング学習の初等的な段階に比較して，より高等的な，プログラムの優れた構成を学び適切な抽象化のための能力を身につける段階を支援する．よって 3D 図形言語は，同じ目的のもとで構成されている SICP の 2D 図形言語を学ぶ課程に沿い，Scheme 上に体系立てられた言語拡張として実装する．

3 要求分析と設計

本節ではオリジナルの図形言語を拡張して 3D 図形を構成，描画し，そのモデルデータを 3D プリント可能な形式へ変換するためのシステムについて，まずその要求される仕様を明確にした上で，それらを満たすよう設計する．本システムの直接的なユーザはこれを使用して 3D 図形を構成し造型しながらプログラミングを学ぶ学生等であり，間接的なユーザはこれを教育のために用いる教育者等である．本稿では本システムを 3D 図形言語と呼ぶことにし，対照のためにオリジナルの図形言語を 2D 図形言語と呼ぶことがある．

我々の 3D 図形言語は主として手続き・データ抽象化をはじめとした基礎的な概念の学習というプログラミング教育の用に供されるものである．その機能として SICP による図形言語の拡張としての 3D 図形の構築と描画の仕組みを持ち，構成された 3D 図形を 3D プリンタにより造型可能なデータ形式へ変換するために OpenSCAD スクリプト形式への変換を可能とすることが求められる．非機能的な面は特に使用性や可搬性について，SICP によるカリキュラムに沿った理解と学習が可能であるよう図形言語の思想に沿い設計されること，Scheme というプログラミング学習のために適した言語の特長を生かしていること，ユーザであるプログラミング学習者の興味をひき積極的に利用しようという意識を持たせうること，異なる計算機環境においても利用が容易であること，実用に不自由ない程度のユーティリティやサンプルが提供されることが必要である．

3D 図形言語は機能の依存関係に基づき，描画のための系統と 3D プリンタによる造型のための系統に大別できる．3D 図形の構築は描画と造型の両者に必要であるが，後段のレンダリングやデータ形式の変換はそれぞれ独立である．

3.1 機能要求

我々は本システムに次の機能を求める．

- 本システムは 3D ペインタを構成する手続きを提供する．2D 図形言語においてペインタの評価が図形の描画を意味していたのと同様に，3D 図形言語における同等のもの，3D ペインタを評価することによって 3D 図形が描画される．3D ペインタはその評価時に，あらかじめ定義された視点と光源の情報に基づき，視点から見た場合の 3D 図形の見え方を計算して像を求め，ウィンドウ上にそれを描画する．
- 本システムは 3D フレームを構成する手続きを提供する．2D 図形言語においてフレームがアフィン変換を表す三次正方行列に等価であったのと同様に，3D 図形言語における同等のもの，3D フレームはアフィン変換を表す四次正方行列に相当する情報を持つ．3D フレームの 3D ペインタへの適用は，3D ペインタの保持する 3D 図

形への 3D フレームの表す変換の適用である．

- 本システムは 3D ペインタに 3D フレームを適用し，新たな 3D ペインタを構成する手続きを提供する．
- 本システムは複数の 3D ペインタを合成し，新たな 3D ペインタを構成する手続きを提供する．
- 本システムは 3D 図形のエクスポート手続きを提供する．エクスポート手続きは 3D ペインタの表す 3D 図形をファイルに書き出す機能を持つ．3D 図形は 3D ペインタ内部において実装上の表現で保持されており，ファイルに書き出す時にデータ形式の変換を行う．出力先のデータ形式は，3D プリント可能な形式へ変換する目的で OpenSCAD スクリプト形式を選択できるものとしテキストファイルとしてこれを生成する．
- 本システムは図形定義のための空間座標系を持ち，その中に存在する図形をウィンドウという平面座標系に投影するために必要な視点と光源を定義する手続きを提供する．
- 本システムは Scheme の言語拡張として構成され，Scheme の機能を包含する．すなわち本システムは Scheme の文法に従い，新たに定義される手続きのほか，Scheme がもともと提供する手続きは全て利用可能であるものとする．
- 本システムは，ユーザのプログラミングや図形言語への理解の程度によって基本的な利用から応用的な利用まで柔軟に活用され，その理解をさらに深めることができる．基本的な利用とは，与えられた座標系上でプリミティブな図形を構成し，またそれらを複合化することでプログラミングの練習となす利用である．応用的な利用とは，自ら座標系を定義するなどして，本システムの提供する基本的な利用の範囲を越えた図形構成をするために，本システムの提供する個々の関数をそのまま，あるいは改変して利用することを指す．

3.2 想定する利用シナリオ

本システムを講義で利用するシナリオは大まかに次のとおりである．

1. ユーザである学生は作成したい 3D 図形を想定し，それを実現するためのプリミティブ図形と変換の組み合わせを考える．
2. この 3D 図形を表現する 3D ペインタを，プリミティブ図形を表す 3D ペインタと 3D フレームの構成手続きを用いながら構成する．
3. 視点及び光源を定義し，項目 2 で構成した 3D ペインタを描画して確認しながら，所望の 3D 図形を作成するために必要に応じて修正を加えるという作業を繰り返す．
4. 完成した 3D ペインタを描画し，この 3D モデルを保存または造型する目的で，ファイルにエクスポートする．

3D プリンタにより造型するためにエクスポートされた 3D モデルのファイルは，OpenSCAD スクリプト形式であり，OpenSCAD を用いて STL 形式等の，3D プリンタの対応するデータ形式へと変換する．変換されたデータを以って 3D プリンタによる 3D モデルの造型を行い，出来上がったモデルを学生に与えてフィードバックとする．

3.3 3D 図形の構成と描画機能の設計

3D ペインタの概念的な定義は上述のとおり，3D 図形の表現を内部に保持し描画する手続きとして与えられる．評価される際には引数として 3D フレームを取り，その表す変換を図形に適用し変形させたものについて，さらに視点と光源の情報に基づいてレンダリングして表示する．なお本稿ではこの 3D ペインタにより保持される，3D 図形の形状や色の情報を持つデータ構造を指して 3D モデルと呼ぶことにする．

ユーザは複合的な 3D ペインタを構成しようとするとき，そのプリミティブな要素となる図形が何であるかを考え，それと変換を組み合わせて複合化していく．よってコンパクトな仕様と高い利便性を両立するためには，システムにより提供されるプリミティブな図形と変換は注意深く選択される必要がある．特に本システムは OpenSCAD 形式への変換機能も有するが，この変換のための実装や計算のコストを減らすためにも，また 3D モデルの構成に特化した同形式の設計を尊重する上でも，OpenSCAD の提供するプリミティブ図形，変換，演算の関数を模倣することが有益

である．よってすべての 3D 図形の構成のための基本となる手続きとして，頂点と面の集合を引数に取って多面体を持つペインタを構成する手続きを提供する．この上で直方体や球といった，プリミティブな図形を構成する手続きを提供する．

3D フレームはアフィン変換を表す三次正方行列と同等の情報を持つ．アフィン変換は図形に対する操作として，拡大縮小，回転，平行移動，剪断等の変形を表すことができる．2D 図形言語と同様に，原点と 3 つの基底ベクトルの変換先の座標を以ってその変換を意味する 3D フレームを構成できるようにする．

ペインタにフレームを適用して得た図形を新たにペインタとして構成する手続きは 2D 図形言語においてはユーザ自らが定義できるものとしているが，本システムではあらかじめこの手続きを提供するものとする．また複数の 3D ペインタを合成する場合，2D 図形言語においては単に塗り重ねればよかったが，3D 図形については座標系と視点の位置関係から前後を解決する必要がある．このため複数の 3D ペインタを引数に取り，それらを合成して新たなペインタとして定義する手続きも提供する必要がある．

また 3D 図形の描画のためにいくつかの補助的な手続きが必要である．まず本システムの想定する右手系の座標系において，3D 図形を特定の位置から特定の方向を向いて見る上で必要な情報を含む視点を構成する手続きがある．次に光源であり，2D 平面上で 3D 図形の立体感を得るための重要な要素である表面の陰影を決定することに利用されるもので，座標系上の位置とその性質によって特徴づけられる．光源の性質とは，例えば発せられる光が並進する平行光源であるか放射される点光源であるかや，強度を表す係数である．光源は複数存在することを許す．また個々の図形の反射等の特性に制限を与えることと光源に色を持たせることは等価であるから，単純化のために光源の色は考えないものとする．さらに 3D 図形表面における色を決定するために，表面属性として反射等の特性をモデルに付加する必要がある．

3.4 エクスポート機能の設計

次に本システムのもう 1 つの主要な部分である，3D モデルのエクスポート機能について設計する．3D モデルは 3D ペインタに保持される 3D 図形の実体であり，このデータを OpenSCAD 形式等へ変換して出力するのが本機能である．

3D ペインタの設計において，個々の図形構成手続き及び変換手続きは OpenSCAD スクリプトの構成を模倣し，そのサブセットとして用意することにした．このため 3D 図形言語と OpenSCAD スクリプト間の手続きレベルでの対応は容易であるものの，複合的な図形を表す 3D ペインタを同スクリプトに変換するためには，その複合的な構造を解析する必要がある．複合的な 3D ペインタは複数のプリミティブな図形についての合成や変換を繰り返し適用することで構成されるから，そのプリミティブな図形を葉に持ち，合成や変換といった操作を節に持つ木構造とみなすことができる．故にこの木構造を根から辿りながら，それぞれの節と葉の要素に対応する適切な OpenSCAD スクリプト関数を選択しながら翻訳する．

また本システムでは，内部表現である S 式による 3D モデルのインポート及びエクスポート機能も提供する．これは単に 3D モデルの保存という意味合いのほかに，実際のモデルの表現を目で見て確認し，構造の理解やデバッグに活用することを可能とする．

3.5 インタフェースの設計

上記を踏まえ，本システムで提供する手続きを表 1 から表 4 に分類する．表 1 は 3D ペインタと 3D フレームの構成のための基礎的な手続きの一覧である．`painter:polyhedron` は表面属性，頂点のリスト，面を意味する頂点のインデックスの三つ組のリストから，多面体を表す 3D ペインタを構成する．`painter:transform` は 3D ペインタ，3D フレームから，変換を適用した図形を持つ新しい 3D ペインタを構成する．`painter:union` は複数の 3D ペインタから，それらを合成した新しい 3D ペインタを構成する．`make-3d-frame` は原点と基底ベクトルの変換先の座標から，3D フレームを構成する．

表 2 は 3D 図形描画のための補助的な手続きの一覧

表 1　3D ペインタ構築のための基本的な手続き

手続き名	引数の型		戻り値の型
(painter:polyhedron	<attribute> <3D vector list> <3D vector list>)	⇒	<3D painter>
(painter:transform	<3D painter> <3D frame>)	⇒	<3D painter>
(painter:union	<3D painter> . <3D painter>s)	⇒	<3D painter>
(make-3d-frame	<3D vector> <3D vector> <3D vector> <3D vector>)	⇒	<3D frame>

表 2　3D 図形の描画のための補助的な手続き

手続き名	引数の型		戻り値の型
(make-camera	<3D vector> <3D vector> <3D vector>)	⇒	<camera>
(make-parallel-light	<3D vector> <intensity>)	⇒	<light>
(make-point-light	<3D vector> <intensity>)	⇒	<light>
(make-intensity	<number> <number> <number>)	⇒	<intensity>
(make-attribute	<color> <color> <color> <color> <number>)	⇒	<attribute>

表 3　プリミティブな図形構成と変換のための手続き

手続き名	引数の型		戻り値の型
(painter:cube	<attribute> <3D vector>)	⇒	<3D painter>
(painter:sphere	<attribute> <number> <number>)	⇒	<3D painter>
(painter:cylinder	<attribute> <number> <number> <number> <number>)	⇒	<3D painter>
(painter:revolution	<attribute> <3D vector list> <number>)	⇒	<3D painter>
(painter:scale	<3D vector> <3D painter>)	⇒	<3D painter>
(painter:rotate	<number> <3D vector> <3D painter>)	⇒	<3D painter>
(painter:translate	<3D vector> <3D painter>)	⇒	<3D painter>

表 4　描画のユーティリティと入出力のための手続き

手続き名	引数の型		戻り値の型
(show	<3D painter> . <3D frame>)	⇒	<boolean>
(redraw	)	⇒	<boolean>
(export	<string> . <symbol>)	⇒	<boolean>
(import	<string>)	⇒	<boolean>

である．make-camera は視点の位置，視線の方向，視野範囲から，視点を構成する．make-parallel-light は光線の方向，光源の強度から，平行光源を構成する．make-point-light は光線の位置，光源の強度から，点光源を構成する．make-intensity は光源の性質を表す係数から，光源の強度を構成する．make-attribute は 2D ペインタの色，環境反射，拡散反射，鏡面反射のそれぞれを表す色，輝度から，表面属性を構成する．

表 3 は表 1 の手続きで定義可能であるが，利便性向

上のためにあらかじめ用意したプリミティブな図形構成と変換のための手続きの一覧である．painter:cube は表面属性，各辺の長さから，直方体を表す 3D ペインタを構成する．painter:sphere は表面属性，半径，ポリゴン分割の解像度から，球体を表す 3D ペインタを構成する．painter:cylinder は表面属性，高さ，上底面の半径，下底面の半径，ポリゴン分割の解像度から，円錐台を表す 3D ペインタを構成する．painter:revolution は表面属性，回転断面の頂点のリストから，回転体を表す 3D ペインタを構成する．painter:scale は各軸方向の倍率，3D ペインタから，それを拡大又は縮小した新しい 3D ペインタを構成する．painter:rotate は回転角度，回転軸，3D ペインタから，それを回転した新しい 3D ペインタを構成する．painter:translate は変位，3D ペインタから，それを平行移動した新しい 3D ペインタを構成する．

表 4 は描画のユーティリティや，モデルの入出力のための手続きの一覧である．show は 3D ペインタ，3D フレームから，その 3D ペインタを評価するもので，3D フレームが省略された場合は恒等変換を表す 3D フレームを 3D ペインタに与えて評価する．すなわち，3D ペインタの表す図形がそのまま描画される．redraw は引数を取らず，直前に描画した 3D 図形を再描画する．export はファイル名，形式を表すシンボルから，直前に描画した 3D 図形のモデルをエクスポートする．形式を表すシンボルは 'SCAD が指定でき，これが指定された場合は OpenSCAD 形式でエクスポートする．シンボルが省略された場合は S 式としてエクスポートする．import はファイル名から，そのファイルに S 式として含まれる 3D モデルをインポートする．

本システムは Scheme 上で実装され，これらの手続きはインタプリタによってインタラクティブに評価し結果を得ることができるものとする．

4 実装

本節では 3D 図形言語の実装について説明する．本システムは Scheme のもともと提供する機能を全て用いることができるという要求に従うため，Scheme

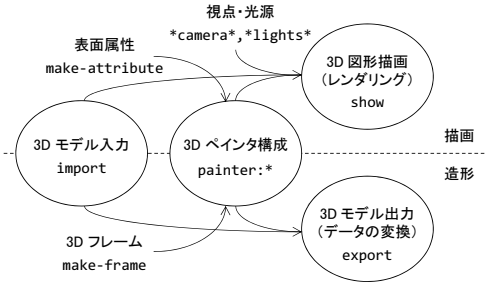


図 4 各機能と主な手続きの対応

上で，2D 図形言語の機能を利用して実装する．これは本システムにより提供される 3D 図形を描画する機能を，2D 図形言語により提供される 2D 図形構築と描画のための手続き及びウィンドウ上への表示のための機能へと還元することを意味する．特に本システムは Scheme 処理系として JAKLD [33] を利用するため JAKLD の提供する 2D のペインタ構成手続きに依存するが，これら手続きはペインタ構成の基本的な概念に従うものであり一般的な性質を有しているから，2D 図形言語を実装している他の処理系への移植も容易であると考える．

以下ではまず 3D ペインタの構成のため，3D モデルや 3D フレーム，その他補助的なデータについての実装を述べる．その上で 3D 図形を描画するレンダリング機能の実装を説明し，さらにその 3D モデルを OpenSCAD スクリプトへ変換してエクスポートする機能の実装を述べる．本システムの機能である構成・描画・入出力によるエクスポートの機能と，実装する主な手続きの対応関係は図 4 に示される．

4.1 3D 図形の構成機能の実装

まず 3D 図形の構成の基本となる，データ構造について説明する．本システムはすべて S 式と呼ばれる式で記述され，最もプリミティブなデータ構造はリストであるなど，Scheme の特徴を受け継いでいる．座標や数学的意味でのベクトルはリストで表現され，行列は列ベクトルからなるリストで表現される．方向ベクトルは単位ベクトルで表す．リストは集合を表すためにも用いられる．

以下では，いくつかのデータ構造を属性リストを用

```
(define (make-model type subtype properties polygons)
  (list 'type type              ; 3D モデルであることを示すシンボル
        'subtype subtype        ; 3D 図形の種類を示すシンボル
        'properties properties ; パラメータ値
        'polygons polygons))    ; ポリゴンの集合
```

図 5　3D モデル構成手続き `make-model`

```
(define (make-painter type subtype properties polygons)
  (let ((model (make-model type subtype properties polygons)))
    (lambda (frame-or-key)
      (case frame-or-key
        ('type type)
        ('model model)
        (else (draw (transform-model model frame-or-key)))))))

(define (painter:polyhedron attribute points triangles)
  (make-painter 'dim3
                'polyhedron
                (list points triangles)
                (make-polygons-polyhedron attribute points triangles)))
```

図 6　3D ペインタ構成手続き `painter:polyhedron`

いて定義する．属性リストは P (Property) リストとも呼ばれ，奇数番目の要素が示す属性の値を直後の偶数番目の要素として格納するリストである．

4.1.1　3D モデルのデータ構造

3D モデルは 3D ペインタ中に保持される 3D 図形の実体であり，3D ペインタの評価時に参照されて描画のためにレンダリング手続き `draw` に渡される．またエクスポート時には，3D 図形を OpenSCAD スクリプト形式によって表す記述を生成するためにエクスポート手続き `export` から参照される．

3D モデルは図 5 に示すように，3D モデルであることを示すシンボル，3D 図形の種類を示すシンボル，形状を決定する上で必要となるパラメータ値，3D 図形の表面の多角形であるポリゴンの集合を要素に持つ属性リストとして実装する．3D 図形の種類を示すシンボルは提供するプリミティブな図形ごとに存在し，その形状を決定するパラメータの種類は各図形の種類に固有のものである．ポリゴンは 3D 図形を多面体として近似して，各面についての形状や明暗を計算し，立体感を表現するもので，多角形を構成する頂点が反時計回りとなる向きを外側として定義する．

ポリゴンは表面属性と頂点座標のリストを要素に持つ属性リストである．表面属性は 2D 図形である場合の色，環境，拡散，鏡面反射の各特性としての色，発光色，輝度の値である (反射特性は 4.2 節で詳説)．色は RGB 値を要素に持つ属性リストである．

4.1.2　3D ペインタのデータ構造

3D ペインタは図 6 に示すようにクロージャとして実装する．3D ペインタを構成する手続きは，3D 図形についての情報を受け取り 3D モデルを構築して，自身の内部のスコープに保持する．さらにこのスコープ内に 3D ペインタを構築することで，それが評価される際には構築された 3D モデルを参照することが可能となる．実際の評価時には，手続き `transform-model` により引数に受け取った 3D フレームの表す変換をこの 3D モデル中の各座標に適用し，変換後の 3D モデルをレンダリング手続き `draw` に渡す．

3D モデルを個々の 3D ペインタのクロージャ内に保持する利点としては，個々の手続き内で完結させることができ，そのペインタの表す 3D モデルの実際のデータを隠蔽してインタフェースと実装とを分離できる点である．また 3D ペインタ自体が不要になれ

```
(define (make-3d-frame o u v w)
  (list 'type 'dim3            ; 3D フレームであることを示すシンボル
        'origin o              ; 原点の変換先
        'edges (list u v w)))  ; 各基底ベクトルの変換先のリスト
```

図 7 3D フレーム構成手続き `make-3d-frame`

ば 3D モデルも参照されずガーベッジコレクタに回収されることが期待でき，3D モデルを 3D ペインタから独立に，例えば大域変数等で集中的に管理する方法に比べて実装の手間を少なくする．

3D ペインタの変換や合成の際には，隠蔽された 3D モデルにアクセスして改変する．すなわち 3D ペインタは 3D フレームのみでなく，メッセージを表すシンボルを受け取り内部に隠蔽された 3D モデルを返す手続きとして実装する．このときメッセージによる 3D モデルの取得操作をユーザから隠蔽するため，3D ペインタを構成する操作は構成手続きによるラップが必要になる．故に 2D 図形言語では任意で定義するものとしていた `painter:transform` も，本システムにおいてはあらかじめ提供する．

4.1.3 3D フレームのデータ構造

3D フレームは図 7 の通り，3D フレームであることを示すシンボル，原点の変換先，各基底ベクトルの変換先のリストを要素に持つ属性リストである．

4.2 3D 図形の描画機能の実装

3D 図形の描画は，それが実際に存在するかのように 2D の画像で表現する処理であり，レンダリングと呼ばれる．レンダリングは 3D ペインタが評価された時，手続き `draw` に 3D モデルが渡されて実行されるもので，投影，クリッピング，シェーディング，陰面・陰線消去，及びこれらを経て得られた処理済みのポリゴンの描画という，5 つの段階に分割できる．以下ではまず 3D 図形の描画に必要となる視点と光源のデータ構造を述べ，次にレンダリングの各段階について説明する．

4.2.1 視点，光源のデータ構造

視点は位置を表す座標，視線の方向ベクトル，視野範囲を要素に持つ属性リストである．視野範囲は視線の方向に対する縦，横，奥行きの幅の値を要素に持つリストであり，この可視範囲に入る図形のみが描画されるものとしている．

光源は平行光源と点光源の種類があり，3D 図形の外側に置かれる．平行光源は，平行光源であることを示すシンボル，光線の方向ベクトル，強度を要素に持つ属性リストである．また点光源は，点光源であることを示すシンボル，光線の位置を表す座標，強度を要素に持つ属性リストである．光源の強度は後述する Phong の反射モデルにおける係数 i_a，i_d，i_s である．

視点と光源は本システム全体に対して効果を持つものであるから，大域変数 `*camera*` と `*lights*` によって管理する．光源は複数存在することを許し，`*lights*` は光源のリストである．これらは 3D モデルのレンダリング時，すなわち 3D ペインタの評価時に参照される．

4.2.2 投影

まず投影は，各ポリゴンの形状の 3D から 2D への変換である．代表的な手法に平行投影と透視投影 [12] があり，前者は視点からの距離によらず大きさが変化しない手法，後者は距離が遠いほど大きさも小さくなる手法である．本システムは，より実際の物の見え方に近い透視投影をデフォルトで採用し，必要に応じて平行投影へ切り替えられるようにする．

4.2.3 クリッピング

次にクリッピングであるが，これは視野範囲外に存在するポリゴンを描画対象から除外し，処理を軽減する段階である．本システムでは視点とともに視野範囲をユーザが定義可能としているから，簡単化のためにその範囲外に存在する頂点を持つポリゴンは全て描画しないものとする．

4.2.4 シェーディング

シェーディングは，光源と物体の表面特性によってポリゴンの色を変化させる．これは物体表面における光の反射をモデル化したシェーディングモデルに

表 5　変換時の手続き・関数の対応

3D 図形言語上の表現	OpenSCAD スクリプト上の表現
(painter:polyhedron <A> <P> <T>)	polyhedron(points=[<P>], triangles=[<T>]);
(painter:union <P>s)	union(){ <P>s; }
(painter:transform <P> <F>)	multmatrix(m=[<F>]){ <P>; }
(painter:cube <A> <V>)	cube(size=[<V>]);
(painter:sphere <A> <R> <S>)	sphere(r=<R>, $fn=<S>);
(painter:cylinder <A> <H> <L> <U> <S>)	cylinder(h=<H>, r1=<L>, r2=<U>, $fn=<S>);
(painter:scale <V> <P>)	scale(v=[<V>]){ <P>; }
(painter:rotate <N> <V> <P>)	rotate(<N>, [<V>]){ <P>; }
(painter:translate <V> <P>)	translate(v=[<V>]){ <P>; }

基づいて行われる．本システムは Phong の反射モデル [24] とフラットシェーディング [12] を採用し，このモデルに従って物体に照射される光線と表面特性に対する物体の色を計算する．Phong の反射モデルは物体表面の色を表す反射光を環境反射光，拡散反射光，鏡面反射光の和として定義する．点 $\mathbf{p} = (x, y, z)^{\mathrm{T}}$ における色 $I_{\mathbf{p}}$ は式 (3) によって表される．

$$I_{\mathbf{p}} = k_a i_a + \sum_{\text{lights}} (k_d i_d \mathbf{l}_{\mathbf{p}}^{\mathrm{T}} \mathbf{n}_{\mathbf{p}} + k_s i_s (\mathbf{r}_{\mathbf{p}}^{\mathrm{T}} \mathbf{v}_{\mathbf{p}})^{\alpha}) \quad (3)$$

ここで k_a，k_d，k_s が環境，拡散，鏡面の各反射光に対する物体表面の特性，i_a，i_d，i_s が光源の強度を表す特性であり，$\mathbf{l}_{\mathbf{p}}$ は $\mathbf{p}$ へ照射する光線の方向，$\mathbf{n}_{\mathbf{p}}$ はポリゴンの法線，$\mathbf{r}_{\mathbf{p}}$ は光線の反射方向，$\mathbf{v}_{\mathbf{p}}$ は $\mathbf{p}$ から視点への方向，そして $\alpha \geq 0$ が輝度である．ポリゴンの法線はポリゴン表面に垂直な外向きの単位ベクトルとして定義する．

4.2.5　陰面・陰線消去

さらに陰面・陰線消去は，ポリゴン間の位置関係をもとに，他のポリゴンに遮蔽されるポリゴンを見掛け上で消去する段階である．本システムは Z ソート法 [12] を採用する．Z ソート法は奥行きソート法とも呼ばれ，スクリーンからの距離の遠いポリゴンを先に描画し，近いものを後に描画して上書きするという，画家のアルゴリズムの具体化である．スクリーン上の画素ごとでなく，ポリゴンごとに描画すればよい Z ソート法は，後段の処理の際にも 2D の多角形描画手続きを利用できるため実装が簡易になり，既存の 2D 図形言語とも親和性が高い．

4.2.6　描画

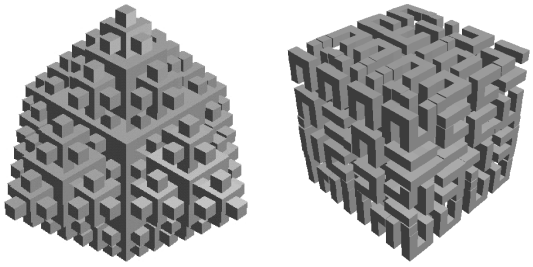
以上の段階を経て得られるデータは，もとの 3D 図形の各ポリゴンがスクリーン上に描画されるべき形状と色に変換され，スクリーンからの距離によって降順に並べられたものである．この変換後の各ポリゴンを描画することでレンダリングを終える．

4.3　エクスポート機能の実装

3D モデルのエクスポート機能は 3D モデルの木構造を再帰的に辿りながら，木構造の各節の意味する変換や合成といった操作，および葉の意味するプリミティブな図形を，それぞれに対応する OpenSCAD スクリプト上の関数に変換する．3D モデルに変換が適用される場合に，子要素としてそのモデル自身を含むように新たな 3D モデルが作られ階層構造をなすため，エクスポート手続きはこの階層構造を取るモデル，すなわち属性リストを外側から内側へ向かって解析するということである．本システム上の図形構成や変換等の手続きと OpenSCAD スクリプトにより提供される各関数との対応関係は表 5 にまとめられる．

5　利用実績とソフトウェアの進化

本節では，3D 図形言語をプログラミング教育の現場で取り入れその有効性を確認するとともに，ソフトウェアの進化という観点から本システムを概観する．以下ではまず本システムを講義の補助教材として利用



(a) フラクタル図形 (b) Hilbert 曲線

図 8 提出を受けた 3D 図形の一部

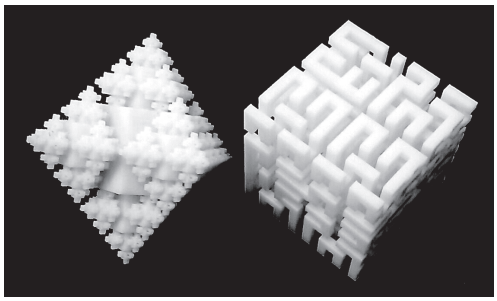


図 9 提出作品を造型したモデル

した実績について述べ，その受講生による反応及び本システムによって作成された 3D モデルの例を挙げて考察する．次いでソースコードの公開やドキュメント整備，デバッグや利便性の改善等について述べる．

5.1 講義における利用と考察

講義「アルゴリズムとデータ構造入門」は学習内容への理解と到達を確認するためいくつかの課題を受講生に課している．2D 図形言語のもとで手続きとデータの抽象化を実践しながら独創的，あるいは芸術的な図形の創作を通して，プログラムのよりよい構造化のためのセンスを身につける学習に役立てるという必修課題はその 1 つである．本研究ではこれに加え，3D 図形言語を用いて独創的な 3D 図形を創作する随意課題を設け，作品とアンケートの提出を受け付けた．提出作品は 3D プリンタを用いて造型し，寸評とともに学生に与えてフィードバックとした．

作品の提出は総数 16 件，アンケートは提出は総数 13 件であった．これらのうち，図 8a に示す独自のフラクタル図形と図 8b に示す Hilbert 曲線 [13] を紹介する．前者は最小単位を立方体とし，親子関係として親の立方体の 6 つの面の中心にそれぞれ子の立方体を接して配置する構造を取る．後者は空間充填曲線というフラクタル図形の一種であり，立体的なコの字型の要素が再帰的に呼び出され，それぞれの間が接続された構造を持つ．それぞれを造型した結果を図 9 に示す．

またアンケートによって 3D 図形言語とその随意課題に対する感想を調査した．これによると「おもしろく，ぜひ挑戦したい」,「3D 画像をテキストベースで描くことができ驚いた」,「Scheme と 3D コンピュータグラフィクスが関連し，さらに 3D プリンタの活用へ発展しているのが興味深い」といった好意的な感想を受け取った．以上から本システムの利用によって，3D 図形の構成を通して手続きとデータ構造の抽象化についての理解を深め，プログラム作成のための基本となる技術力の向上が期待できる．また画像という目に見える結果を出力し，3D プリンタにより手に取ることのできるモデルを造型することで，より学生の興味をひき，講義及びプログラミング学習に対する積極性を増すことが可能である．

5.2 公開と利便性向上の取組

本研究の成果物である 3D 図形言語のソースコードは Github †1上，講義に用いたスライドは SlideShare †2 上で公開しており，誰でも閲覧し利用することが可能である．また本システムは毎年の講義における受講生や教員からの反応を受けてアップデートやデバッグを行うものであり，常にその利便性向上に向けた取組がなされている．

6 結論

本研究は講義の補助教材として，Scheme 上で 2D 図形を構成・描画するシステムである図形言語を拡張し，3D 図形の構成・描画と 3D プリンタによる造型を可能とするシステム 3D 図形言語を開発した．対応する図形を目で見て手に取って確認し，プログラム構築のための基盤的な素養としての手続き抽象化

†1 https://github.com/vi-iv/jakld-3dcg

†2 http://www.slideshare.net/vi-iv/jakld-3dcg

とデータ抽象化の理解を支援すること，および学生の講義内容の積極的な学習を促すことを目的とした．方針としてオリジナルの図形言語から連続的に学ぶことができるよう統合性に注意し，初学者のプログラミング学習に適するという Scheme の特長を生かしつつ，CSG による構成的な 3D 図形作成のための仕組みを取り入れた．3D 図形の構成と描画の機能に加え OpenSCAD スクリプト形式への変換機能を持たせることで，ユーザの構成した 3D 図形を 3D プリンタにより造型して実体化し，プログラムの構造に対する視覚的かつ直観的な理解を可能とした．また本システムを実際に講義で使用し，課題の提出やアンケートによって学生からの良好な反応を受け取って有効性を確認した．

本システムは拡張性にも優れており，様々な拡張が機能面，非機能面の両方において考えられる．例えば 3D フレームはアフィン変換と同等の変換しか実現できないが，多面体ペインタの構成手続きに与える頂点を求める段階での体系化により，射影変換のような変換や双曲座標系での図形の定義も可能となる．また OpenSCAD スクリプトで提供されている複数の 3D 図形の共通部分や差集合を取る関数を実装することで図形構成の自由度を向上できる．よりインタラクティブな GUI を実現するため，3D 図形の連続の表示によるアニメーションの実現や，描画機能としてレンダリングエンジンを実装するのではなく，処理系の拡張や外部のツールの利用が可能であるか探ることも考えられる．本システムの教育上の利用としては，そのチュートリアル的構成や，目標に掲げる手続き・データ抽象化をはじめとするプログラミング基礎理解についての実際の到達度の評価に改善の余地があり，今後の利用とユーザによるフィードバックを通してソフトウェアの進化の一層の促進を目指していく．

謝辞 本研究の一部は，科研費 24220006 の支援を受けた．また本稿中の図 8a 及び図 8b として紹介した 3D 図形は，それぞれ受講生の枡井啓貴君と豊島悠紀夫君による作品である．

参 考 文 献

[1] 3D Systems, Inc.: Stereolithography Interface Specification, Technical report, 3D Systems, Inc., Valencia, CA, USA, 1988.

[2] Ableson, H., Dybvig, R. K., Haynes, C. T., Rozas, G. J., Adams IV, N. I., Friedman, D. P., Kohlbecker, E., Steele Jr., G. L., Bartley, D. H., Halstead, R., Oxley, D., Sussman, G. J., Brooks, G., Hanson, C., Pitman, K. M. and Wand, M.: The Revised5 Report on the Algorithmic Language Scheme, *Higher-Order and Symbolic Computation*, Vol. 11, No. 1(1998), pp. 7–105.

[3] Ableson, H., Sussman, G. J. and Sussman, J.: *Structure and Interpretation of Computer Programs*, second edition, The MIT press, Cambridge, MA, USA, 1996.

[4] Berman, A. M.: Does Scheme Enhance an Introductory Programming Course?: Some Preliminary Empirical Results, *SIGPLAN Notices*, Vol. 29, No. 2(1994), pp. 44–48.

[5] Bottoni, P., Meyer, B., Marriott, K. and Parisi-Presicce, F.: Deductive Parsing of Visual Languages, in *Proceedings of the 4th International Conference on Logical Aspects of Computational Linguistics*, London, UK, UK, Springer-Verlag, 2001, pp. 79–94.

[6] Dybvig, R. K.: *The Scheme Programming Language*, fourth edition, The MIT Press, Cambridge, MA, USA, 2009.

[7] Evans, B.: *Practical 3D Printers: The Science and Art of 3D Printing*, Apress, Berkely, CA, USA, 2012.

[8] Ferguson, I.: *The Schemer's guide*, second edition, Schemers Inc., Fort Lauderdale, FL, USA, 1997.

[9] Flatt, M., Findler, R. B. and PLT: Guide: PLT Scheme, Technical report, PLT Scheme Inc., 2010.

[10] Hanson, C.: MIT Scheme Reference Manual, Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1991.

[11] Harvey, B.: *Computer Science Logo Style*, The MIT Press, Cambridge, MA, USA, 1997.

[12] Hearn, D. and Baker, M. P.: *Computer Graphics*, Prentice Hall, Upper Saddle River, NJ, USA, 1994.

[13] Hilbert, D.: Über die stetige Abbildung einer Linie auf ein Flächenstück, *Mathematische Annalen*, Vol. 38, No. 3(1891), pp. 459–460.

[14] Jain, A. K.: *Fundamentals of Digital Image Processing*, Prentice Hall, Upper Saddle River, NJ, USA, 1988.

[15] 金田泰: 3D プリンタによる “3 次元タートル・グラフィクス”, 情報処理学会夏のプログラミング・シンポジウム, 東京都, 日本, 情報処理学会, 2014.

[16] 経済産業省大臣官房政策評価広報課: 3D PRINTER にできること, 経済産業ジャーナル, 2013. http://www.meti.go.jp/publication/data/2013_08.html.

[17] Kintel, M. and Wolf, C.: OpenSCAD—The Pro-

grammers Solid 3D CAD Modeller, Accessed: October 2014. http://www.openscad.org/.
[18] Lakin, F.: Visual Grammars for Visual Languages, in *Proceedings of the Sixth National Conference on Artificial Intelligence*, Menlo Park, CA, USA, AAAI Press, 1987, pp. 683–688.
[19] Lakin, F. H.: Computing with Text-graphic Forms, in *Proceedings of the 1980 ACM Conference on LISP and Functional Programming*, New York, NY, USA, ACM, 1980, pp. 100–106.
[20] Majewskia, M.: A tutorial on the realistic visualization of 3D Sierpinski fractals, *Computers & Graphics*, Vol. 22, No. 1(1998), pp. 129–142.
[21] McCarthy, J.: Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I, *Communications of the ACM*, Vol. 3, No. 4(1960), pp. 184–195.
[22] Moon, D. A.: MacLisp Reference Manual, Technical report, MIT Project MAC, Cambridge, MA, USA, 1975.
[23] 奥乃博, 馬谷誠二, 糸山克寿: Syllabus—京都大学工学部, 2013. http://www.t.kyoto-u.ac.jp/syllabus-s/?mode=subject&lang=ja&year=2013&b=6&c=91150.
[24] Phong, B. T.: Illumination for computer generated pictures, *Communications of the ACM*, Vol. 18, No. 6(1975), pp. 311–317.
[25] Requicha, A. A. and Voelcker, H. B.: Constructive Solid Geometry, Technical report, University of Rochester, Rochester, NY, USA, 1977.
[26] Reynolds, C. W.: Computer Animation with Scripts and Actors, *SIGGRAPH Computer Graphics*, Vol. 16, No. 3(1982), pp. 289–296.
[27] Riehl, A. M., Friedman, D. P., Harvey, B., Kaplan, S., Salter, R. and Springer, G.: Using SCHEME in the Introductory Computer Science Curriculum (Abstract), *SIGCSE Bulletin*, Vol. 25, No. 1(1993), p. 288.
[28] Schemers Inc.: Schemers Inc., Accessed: October 2014, http://www.eimacs.com/schemers.htm.
[29] Shiro, K.: Gauche—A Scheme Implementation, Accessed: October 2014, http://practical-scheme.net/gauche/.
[30] Sierpiński, W.: Sur une courbe dont tout point est un point de ramification, *Comptes Rendus hebdomadaires des séance de l'Académie des Sciences*, Vol. 160, No. 1(1915), pp. 302–305.
[31] Springer, G. and Friedman, D. P.: *Scheme and the Art of Programming*, The MIT Press, Cambridge, MA, USA, 1989.
[32] 和田英一: Haskell プログラミング 関数画家, 情報処理, Vol. 46, No. 10(2005), pp. 1163–1171.
[33] 湯淺太一: Java アプリケーション組み込み用の Lisp ドライバ, 情報処理学会論文誌. プログラミング, Vol. 44, No. 4(2003), pp. 1–16.
[34] 湯淺太一, 奥乃博, 尾形哲也: 京大における Lisp を使ったプログラミング教育, 情報処理, Vol. 52, No. 9(2011), pp. 1191–1194.

付録: プログラム例

1 作例 (図 1b) の Sierpiński の正四面体 [20]

```
(define (painter:tetrahedron attribute origin size)
  "再帰の最小単位である正四面体を表す␣3D␣ペインタを構成"
  (let ((s0 (/ 1.0 2.0))
        (s1 (/ 1.0 (* 2.0 (sqrt 2.0)))))
    (let (;; 頂点のリスト
          (ps (list (list     s0 0.0 s1)
                    (list (- s0) 0.0 s1)
                    (list 0.0     s0 (- s1))
                    (list 0.0 (- s0) (- s1))))
          ;; 面を意味する頂点のインデックスの三つ組のリスト
          (ts (list (list 0 1 3)
                    (list 0 2 1)
                    (list 0 3 2)
                    (list 1 2 3))))
      (painter:polyhedron attribute
                          (map (lambda (p) (add (scl size p) origin)) ps)
                          ts))))

(define (painter:sierpinski-tetrahedron attribute size n)
  "Sierpinski␣の正四面体フラクタルを表す␣3D␣ペインタを構成"
  (define (sierpinski-tetrahedron% origin size counter)
    (if (<= counter 0)
        ;; 再帰呼出しの終わりに最小単位の正四面体を配置
        (painter:tetrahedron attribute origin (* 1.3 size)) ; need overlap
        ;; 再帰呼出しが終わりでなければ子要素を配置
        (let ((s0 (/ 1.0 4.0))
              (s1 (/ 1.0 (* 4.0 (sqrt 2.0)))))
          (let ((o0 (add (scl size (list     s0 0.0 s1)) origin))
                (o1 (add (scl size (list (- s0) 0.0 s1)) origin))
                (o2 (add (scl size (list 0.0     s0 (- s1))) origin))
                (o3 (add (scl size (list 0.0 (- s0) (- s1))) origin))
                (s/2 (/ size 2)))
            ;; 正四面体の中心と各頂点の中点の位置に半分の大きさの子要素を配置
            (painter:union (sierpinski-tetrahedron% o0 s/2 (1- counter))
                           (sierpinski-tetrahedron% o1 s/2 (1- counter))
                           (sierpinski-tetrahedron% o2 s/2 (1- counter))
                           (sierpinski-tetrahedron% o3 s/2 (1- counter)))))))
  (sierpinski-tetrahedron% (list 0.0 0.0 0.0) size n))

(define sierpinski-tetrahedron
  (painter:sierpinski-tetrahedron *attribute* *size* *n*))
```

2 提出作品 (図 **8a**) のフラクタル図形

```
(define (painter:cube-center attribute origin size)
  "任意の座標に中心を持つ立方体を表す␣3D␣ペインタを構成"
  (let ((s/2 (/ size 2)))
    (painter:translate
     origin
     (painter:translate
      (map - (list s/2 s/2 s/2))
      (painter:cube attribute (list size size size))))))

(define (painter:too-many-cubes attribute size n)
  "立方体の各面に子要素を持つフラクタルを表す␣3D␣ペインタを構成"
  (define (too-many-cubes% origin size counter)
    (if (<= counter 0)
        ;; 再帰呼出しの終わりに最小単位の立方体を配置
        (painter:cube-center attribute origin size)
        ;; 再帰呼出しが終わりでなければ子要素を配置
        (let ((s/2 (/ size 2))
              (x (+ (/ size 2) (/ size 4))))
          (let (;; 子要素の座標
                (t0 (add origin (list x 0 0)))
                (t1 (add origin (list 0 x 0)))
                (t2 (add origin (list 0 0 x)))
                (t3 (add origin (list (- x) 0 0)))
                (t4 (add origin (list 0 (- x) 0)))
                (t5 (add origin (list 0 0 (- x)))))
            ;; 立方体の各面に半分の大きさの子要素を配置
            (painter:union (painter:cube-center attribute-001 origin size)
                           (too-many-cubes% t0 s/2 (1- counter))
                           (too-many-cubes% t1 s/2 (1- counter))
                           (too-many-cubes% t2 s/2 (1- counter))
                           (too-many-cubes% t3 s/2 (1- counter))
                           (too-many-cubes% t4 s/2 (1- counter))
                           (too-many-cubes% t5 s/2 (1- counter)))))))
  (too-many-cubes% (list 0 0 0) size n))

(define too-many-cubes
  (painter:too-many-cubes *attribute* *size* *n*))
```

3 提出作品 (図 8b) の Hilbert 曲線

```
(define (painter:hilbert-curve attribute size n)
  "Hilbert␣曲線フラクタルを表す␣3D␣ペインタを構成"
  (define (cube% size)
    (painter:cube-center attribute '(0 0 0) size))
  (define (s% p)
    (painter:scale (list 0.5 0.5 0.5) p))
  (define (r% a i p)
    (painter:rotate a (list-ref '((1 0 0) (0 1 0) (0 0 1)) i) p))
  (define (t% v s p)
    (painter:translate (scl (/ s 2) v) p))
  (define (hilbert-curve% counter)
    (if (<= counter 1)
        ;; 再帰呼出しの終わりに最小単位の立方体を配置
        (cube% size)
        ;; 再帰呼出しが終わりでなければ子要素を配置
        (let ((c (1- counter))
              (s (* 2 size (expt 0.5 counter)))
              (l (* (- 1 (expt 0.5 (1- counter))) size)))
          (painter:union
           ;; 立方体を分割したときの隅の部分に子要素を回転して配置
           (t% '( 1  1  1) size (s% (r%  90 1 (hilbert-curve% c))))
           (t% '( 1  1 -1) size (s% (r% 270 1 (hilbert-curve% c))))
           (t% '( 1 -1  1) size (s% (r% 180 0 (r%  90 2 (hilbert-curve% c)))))
           (t% '(-1  1  1) size (s% (r%  90 1 (hilbert-curve% c))))
           (t% '( 1 -1 -1) size (s% (r% 180 1 (r% 270 2 (hilbert-curve% c)))))
           (t% '(-1  1 -1) size (s% (r% 270 1 (hilbert-curve% c))))
           (t% '(-1 -1  1) size (s% (r%  90 0 (r% 270 2 (hilbert-curve% c)))))
           (t% '(-1 -1 -1) size (s% (r% 270 0 (r% 270 1 (hilbert-curve% c)))))
           ;; 各部分の間の曲線を接続する立方体を配置
           (painter:translate (list 0 s l) (cube% s))
           (painter:translate (list 0 s (- l)) (cube% s))
           (painter:translate (list l 0 l) (cube% s))
           (painter:translate (list l 0 (- l)) (cube% s))
           (painter:translate (list l (- s) 0) (cube% s))
           (painter:translate (list (- l) 0 l) (cube% s))
           (painter:translate (list (- l) 0 (- l)) (cube% s))))))
  (hilbert-curve% (1+ n)))

(define hilbert-curve
  (painter:hilbert-curve *attribute* *size* *n*))
```

古川孝太郎

2013 年京都大学工学部情報学科卒業. 2015 年同大学大学院情報学研究科知能情報学専攻修士課程修了. 在学中は音環境理解の研究, プログラミング教材の開発に従事.

糸山克寿

2006 年京都大学工学部情報学科卒業. 2011 年同大学大学院情報学研究科知能情報学専攻博士後期程修了. 同年より同大学情報学研究科知能情報学専攻助教. 京都大学博士 (情報学). 音楽情報処理, 音楽鑑賞インタフェース等の研究に従事.

吉井和佳

2008 年京都大学大学院情報学研究科知能情報学専攻博士後期程修了. 同年より産業技術総合研究所情報技術研究部門研究員. 2014 年より京都大学大学院情報学研究科知能情報学専攻講師. 京都大学博士 (情報学). 統計的機械学習技術に基づく音楽情報処理の研究に従事. 情報処理学会山下記念研究賞, 船井研究奨励賞, ISMIR2013 Best Oral Presentation Award 等受賞.

奥乃 博

1972 年東京大学教養学部基礎科学科卒業. 日本電信電話公社, NTT, JST, 東京理科大学, 京都大学大学院を経て早稲田大学大学院創造理工学研究科教授. 京都大学大学院情報学研究科知能情報学専攻名誉教授. 東京大学博士 (工学). この間, スタンフォード大学客員研究員, 東京大学工学部客員助教授. 人工知能, 音環境理解, ロボット聴覚, 音楽情報処理の研究に従事. 人工知能学会論文賞, 船井情報科学振興賞, IEEE/RSJ IROS-2010 NTF Award for Entertainment Robots and Systems 等受賞.