Scheme による 3D 図形の構成的制作

古川 孝太郎　坂東 宜昭　糸山 克寿　吉井 和佳　奥乃 博

本稿では, LISP 系言語の一つである 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などの抽象的な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感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フレームワーク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結果を報告する.

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」[25] のカリキュラムを改善し, 3D 図形, 特にフラクタル図形の構成と造型を通した, 手続きとデータ構造の抽象化への理解の促進を目的とする. 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フレームワークである図形言語 [3] を取り扱い, これに 3D 図形の構成と描画, 3D プリンタによる造型のための拡張を加える. 実現したフレームワークを講義で取り入れ, 学生から随意課題として作品の提出を受けて, それを造型したモデルとともに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.†1

「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, プログラミングの本質である手続き抽象化とデータ抽象化を学ぶことのできる講義である [24]. 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. このカリキュラムに含まれる図形言語は, 「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することが可能であるため, 手続きの呼出しを視覚的に捉え, 再帰呼出しや階層化等の概念を直観的に学ぶことができる [21].

Scheme は LISP [13] 方言の一つであり, 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 [5,7,19] や処理系 [8,9,17] 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 [4,11,16]. 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 [23] を使用する.

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も提供する. 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

Koutarou Furukawa, Yoshiaki Bando, 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.

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

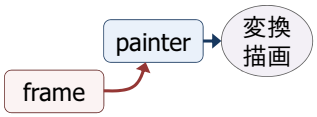


図 1: ペインタの評価

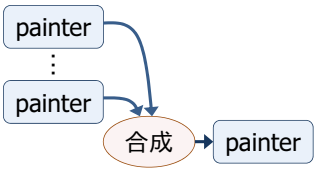


図 2: ペインタの順次評価による合成

めの主要なツールとして広く利用されている [6, 22]. 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 節で図形言語の概要について述べ, 3 節で本研究による図形言語の概念的拡張と実装, 実現されたフレームワークの機能を説明する. さらに 4 節で実際の講義における利用を取り上げ, その結果を考察したのち, 5 節でまとめる.

2 図形言語の概要

本章では, 図形言語の概要について述べる. 図形言語はペインタとフレームという二つの概念から構成される. ペインタとは図形そのものであり, フレームとともに評価することで図形を描画できる (図 1). また複数のペインタを順に評価することで, 図形を塗り重ね合成することが可能である (図 2).

フレームとはペインタに対する座標変換である. すなわちペインタが評価される際, フレームは描画領域を表すものとしてペインタの引数になり, ペインタの表す図形はこの領域に合わせて変換され描画される. フレームは一次変換を表す二次正方行列と平行移動を表す二次ベクトルの対と等価な情報を持つ. それらを $\mathbf{A}, \mathbf{b}$ とすると, フレームによる変換は図形の各点が式 (1) により移されることに等しい (図 3).

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

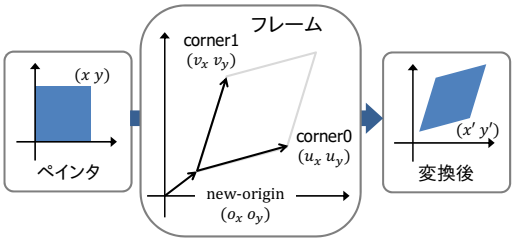


図 3: ペインタへのフレームの適用

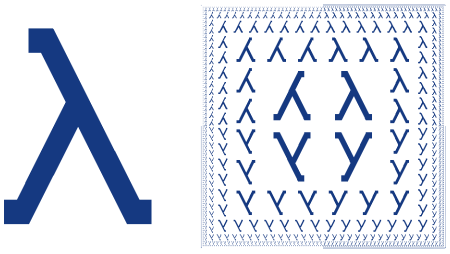
具体的には次のように図形を構成することができる. まずフレームを定義する.

```
(define *frame*
  (make-frame (make-vect 0.0 0.0)
              (make-vect 1.0 0.0)
              (make-vect 0.0 1.0)))
```

ここで `make-vect` は引数に座標を取り点の内部表現を構成する手続きである. `make-frame` はフレームを構成する手続きであり, 描画領域としての座標系の原点と二軸上の各点を引数に取る. 次に頂点の情報から線分や多角形のペインタを構成する.

```
(define *letterlambda*
  (vertexes->painter
    (list (make-vect 0.45 0.60)
          (make-vect 0.25 0.20)
          (make-vect 0.20 0.20)
          (make-vect 0.20 0.10)
          (make-vect 0.30 0.10)
          (make-vect 0.50 0.50)
          (make-vect 0.70 0.10)
          (make-vect 0.80 0.10)
          (make-vect 0.80 0.20)
          (make-vect 0.75 0.20)
          (make-vect 0.40 0.90)
          (make-vect 0.30 0.90)
          (make-vect 0.30 0.80)
          (make-vect 0.35 0.80))
    #t))
```

`vertexes->painter` は JAKLD の提供する手続きであり, 頂点のリストから多角形ペインタを構成する. 第二引数は塗りつぶしの有無である. ここでペインタ `*letterlambda*` にフレーム `*frame*` を与えて評価すると, `*letterlambda*` の表す図 4 の図形が描画される. このフレームワーク上で再帰的にペインタを変換し描画する手続きを定義すれば, square-limit と呼ばれるフラクタル図形が描画できる.



(a) `*letterlambda*` (b) square-limit

図 4: ペインタの再帰的描画

3 フレームワーク拡張の概要

本研究で実現するフレームワークは二つの機能からなる. 一つは既存の図形言語を 3D へ拡張して図形を構成, 描画可能とする機能であり, もう一つは構成された図形データを 3D プリンタの対応する形式へと変換して出力する機能である. 以下の小節ではまず 3D コンピュータグラフィクスの導入と, 図形言語の 3D への拡張, さらに 3D プリンタのためのインタフェースの詳細を順に述べる.

3.1 3D コンピュータグラフィクスの実装

3D コンピュータグラフィクスとは仮想的な立体物を定義し, 現実に存在するかのように画像として表示する技術である. この実現のためには一般に 3D モデルの構成 (モデリング) と, 描画 (レンダリング) の二つの機能が必要であり, 本稿では前者について, フレームワークの提供する手続きの記述から 3D モデルデータを構成する機能と定義し, 後者について, その 3D モデルデータを 2D 画像へと変換する機能と定義する (図 5). また 3D モデルは多面体であるとし, その表面の多角形 (ポリゴン) の集合によって構成されるものとする. 例えば図 6 に示すように, 仮想的な直方体を定義する手続き `make-cuboid` にその立方体の座標や色などの情報を与えたとき, 直方体を構成する六面の長方形のポリゴン集合が得られるモデリング機能と, その各長方形のポリゴンを, 視点の位置と方向によって定義される平面 (スクリーン) 上に投影して得られる図形を描画するレンダリング機能を実現する.

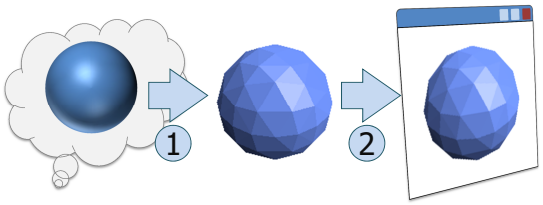


図 5: (1) モデリング, (2) レンダリング

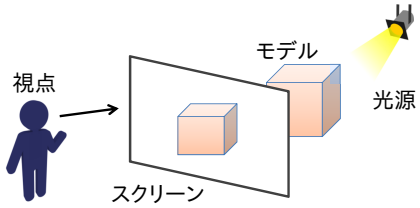


図 6: モデルのスクリーン上の像を計算

モデリングはいくつかのプリミティブな図形を構成する手続きを提供し, その組合せによって多様な図形を構成できるようにする. プリミティブな図形は引数として与えられた情報から多面体の頂点を計算し, それらを結ぶ多角形の集合を作ることによってモデルを与える. この際上図に示すように, 球体や回転体などの曲面を持つ図形についても表面上の点を適当にサンプリングし, その点を結んだ三角形の面からなる多面体に近似する.

レンダリングは図 7 の五つの段階に分割できる. 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 [10] があり, 前者は視点からの距離によらず大きさの変化しない手法, 後者は距離が遠いほど大きさも小さくなる手法である (図 8). 本研究ではより実際の物の見え方に近い透視投影を採用し, 必要に応じて平行投影へ切り替えられるようにした.

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除き, 処理を軽減する段階である. 本研究では視点とともに視野範囲をユーザが定義可能とし, 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した.

シェーディングは, 光の当たり方と物体の表面特性によってポリゴンの色を変化させる段階である. これ

図 7: レンダリングの五つの段階

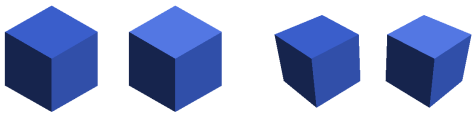
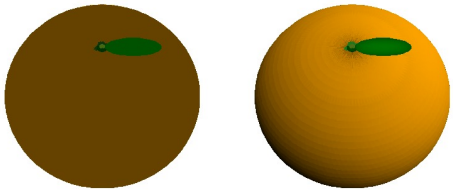
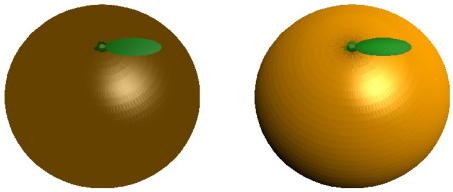


図 8: 平行投影と透視投影



(a) 環境反射のみ　(b) 環境反射と拡散反射



(c) 環境反射と鏡面反射　(d) すべての反射

図 9: Phong の反射モデルにおける各反射光

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基づいて行われる処理である. 本研究では Phong の反射モデル [14] 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 (2) によって表される.

$$I_{\mathbf{p}} = k_a i_a + \sum_{\text{lights}} (k_d i_d \mathbf{l}_{\mathbf{p}}^{\mathrm{T}} \mathbf{n}_{\mathbf{p}} + k_s i_s (\mathbf{r}_{\mathbf{p}}^{\mathrm{T}} \mathbf{v}_{\mathbf{p}})^{\alpha}) \qquad (2)$$

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 α が輝度である. 各反射光の成分がモデルの色に寄与する様子を図 9 に示す.

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. いくつかの代表的な手法があるが, 本研究では Z ソート法 [10] 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.

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.

3.2 図形言語の 3D 拡張

我々はモデリングをペインタ定義時に, レンダリングをペインタ評価時に機能するものとして実装する. モデルはペインタの閉包内で保持しており, ペインタを評価した時に図 10 に示すようにあらかじめ定義された光源や視点, スクリーンをもとにして 3D 図形がどのように描画されるべきかを計算し, `segments->painter` や `vertexes->painter` などの手続きを用いスクリーン上の像を描画する. レンダリングのみペインタ評価時に行うことで, ペインタの構成とそのフレームによる変形のための手続きを容易に階層化でき, 図形言語を自然に拡張することが可能となる. 以下では 3D モデルを表すペインタを 3D ペインタと呼ぶものとする.

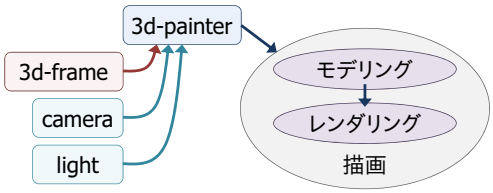


図 10: 3D ペインタ評価時の処理

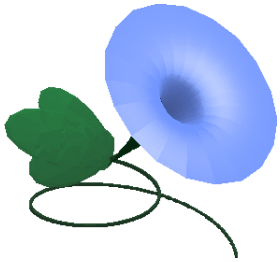


図 11: 3D ペインタの作例

3D ペインタ構成のために三つの手続きを提供する. 多面体を構成する painter:polyhedron とペインタにフレームを適用する painter:transform , 複数の 3D ペインタの和を取る painter:union である. painter:polyhedron は引数としてポリゴンの反射特性と頂点および面についての情報を取って多面体を描画する 3D ペインタを構成する. painter:transform は引数としてペインタとフレームを取ってペインタを変換する. 3D 図形言語におけるフレームは一次変換を表す三次正方行列と平行移動を表す三次ベクトルからなり, それぞれ $\mathbf{A}, \mathbf{b}$ とすると式 (3) によって表すことができる.

$$(x', y', z')^{\mathrm{T}} = \mathbf{A}(x, y, z)^{\mathrm{T}} + \mathbf{b} \tag{3}$$

painter:union は引数にペインタを取って, それらのポリゴンの集合和を持つ新しいペインタを返す.

上記三つの手続きを組み合わせることよって, 任意の 3D モデルを構成できる (図 11). 一方で基本的な図形, 例えば直方体や球といった形状はあらかじめペインタ構成手続きとして定義しておくことで, ユーザはより本質的な部分の記述に注力できるようになる. よって我々は直方体 painter:cube, 球 painter:sphere, 円柱または円錐 painter:cylinder, 回転体 painter:revolution を表すモデルを持ったペインタを得るための手続きを定義する. 同様に基本的な変換, 例えば平行移動や拡大縮小といった変換もあらかじめ定義しておく. これは平行移動 painter:translate, 拡大または縮小 painter:scale, 回転 painter:rotate を表す手続きである.

これらの手続きで具体的には次のようにして任意のペインタを構成する. まず視点と光源を定義する.

```
(define *camera*
  (make-camera '(12.0 12.0 12.0)
               '(-1.0 -1.0 -1.0)
               '(60.0 60.0 200.0)))

(define *lights*
  (list (make-parallel-light
         '(-1.0 -1.0 -1.0)
         (make-intensity 0.5 0.5 0.5))))
```

ここで make-camera の第一引数は視点の位置, 第二引数は方向, 第三引数は各軸方向についての視野範囲である. make-parallel-light は平行光源を構成する手続きであり, 光源の位置と特性を引数に取る. make-intensity は引数に環境反射, 拡散反射, 鏡面反射の度合いを表す係数を取り, 光源の特性を返す. 次にポリゴン表面の反射特性を定義する.

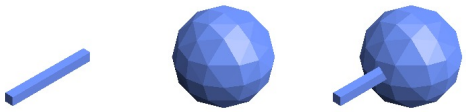
```
(define *attribute*
  (make-attribute (hex->color #x000000)
                  (hex->color #x4169E1)
                  (hex->color #x4169E1)
                  (hex->color #xffffff)
                  (hex->color #x000000)
                  3))
```

反射特性は, 無光源状態での色, 環境反射, 拡散反射, 鏡面反射の各色, 発光色, および輝度を引数に取る make-attribute によって定義できる. hex->color は十六進数の色コードを内部表現に変換する手続きである. この条件の下でペインタを定義する.

```
(define *cube*
  (painter:cube *attribute*
                '(2.0 2.0 20.0)))

(define *sphere*
  (painter:sphere *attribute* 10.0 2))

(define *union*
  (painter:union *cube* *sphere*))
```



(a) `*cube*` (b) `*sphere*` (c) `*union*`

図 12: ペインタ構成の例

`*cube*` は直方体を表すペインタであり, 各辺が 2.0, 2.0, 20.0 の長さを持つ直方体を原点の位置に構成している. `*sphere*` は球を表すペインタであり, 半径 10.0 の球を原点位置に構成する. 第三引数の 2.0 は曲面のポリゴン分割の細かさを表す値である. そして `*union*` は二つのペインタを合成したペインタである. これらのペインタを評価した結果が図 12 である. 特に `*union*` では `painter:union` によってポリゴン間の前後関係が解決されていることがわかる.

同様にしてフラクタル図形を構成する. フラクタル図形の構成には最小単位をなす図形と親子関係という二点に注目する必要がある. 最小単位をなす図形とは再帰呼出しが上限回数に達した際に呼び出される図形であり, 親子関係とは再帰呼出しする側の手続きと, される側の手続きの間の関係である. 例として Sierpinski の三角錐 [18] と呼ばれるフラクタル図形を考える. まず最小単位を三角錐とし, 親子関係は親の三角錐に対してその四つの頂点の位置に子供の三角錐が配置されるものとする. 構成された結果を図 13 に示す. 以上のように図形言語を拡張することで, 階層的, 再帰的な 3D 図形の構成が可能となる.

3.3 3D プリンタのためのインタフェース

手に取れるモデルとして手続きとデータの抽象化や階層化への理解を促すため, 構成した 3D モデルデータを 3D プリンタによって造型するインタフェースを提供する. これは Scheme 上での 3D モデルデータをオープンソースソフトウェア OpenSCAD [12] のスクリプトへと変換するものであり, 同ソフトウェアを用いてデファクトスタンダードなモデル形式の STL (STereoLithography) [1] データへ変換する. 著者らの研究室では 3D プリンタとして uPrint SE [20] を導入しており, STL データからモデルを造型できる.

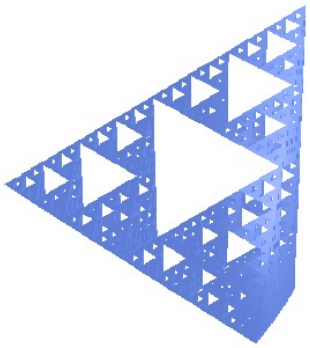


図 13: Sierpinski の三角錐の構成

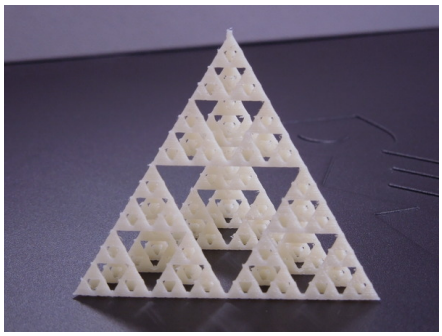


図 14: Sierpinski の三角錐の造型

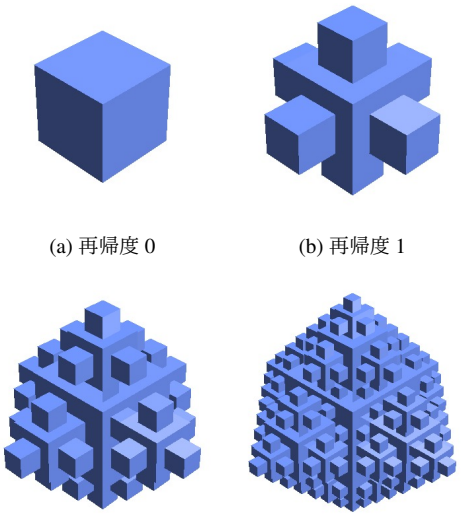
本フレームワークで提供する手続きは, 同様の機能を持つ関数が OpenSCAD スクリプトでも提供されており, 3D モデルの内部表現を対応する関数へ変換するだけでよい. 直前に描画したペインタについて, 以下の手続きを評価するとファイルとして出力される.

```
(export "exported.scad" 'scad)
```

変換後のファイルを用いて 3D プリンタでモデルを造型する. 上記の Sierpinski の三角錐を造型した結果を図 14 に示す.

4 3D 図形言語の講義への導入

本節では実現した 3D 図形言語を講義に用いた概要を述べる. 本講義では毎年学生に 2D 図形言語の作品提出を必須課題として課している. これに加え随意課題として, 本フレームワークにより作成した 3D モデ



(a) 再帰度 0　(b) 再帰度 1

(c) 再帰度 2　(d) 再帰度 3

図 15: 提出作品のフラクタル図形

ルの提出を受け付けた. 提出を受けた図形はフラクタル, 空間充填曲線, ダイヤモンド様の形状など様々であり, それぞれについてデータを変換し, 3D プリンタでモデルを造型した.

このうち図 15 のフラクタル図形を紹介する. これは最小単位を立方体とし, 親子関係として親の立方体の六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. すなわち再帰度が 0 の場合は立方体であり, 1 の場合は大小の立方体がそれぞれ 1 個と 6 個, 2 の場合は大中小の立方体がそれぞれ 1 個, 6 個, 30 個というように, 立方体が増加していく. 造型した結果は図 15 である.

また講義に対するアンケートにおいて, 「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」といった多数の好意的な感想や, 「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」という旨の, 3D コンピュータグラフィクスの黎明期に LISP が利用されていた経緯 [15] を意識しているであろう感想を受け取った. 以上から本フレームワークの利用により, 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と言える. 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, より学生の興味を惹き講義に対する積極性を増すことができた.

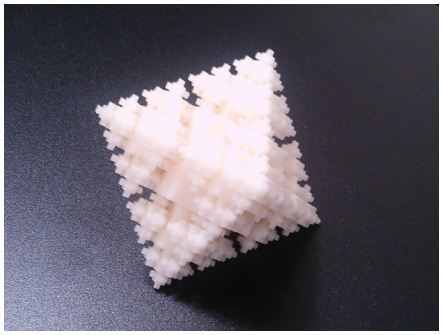


図 16: 造型した作品

5 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, 描画するフレームワークである図形言語を拡張し, 3D 図形の構成と 3D プリンタによる造型を可能とした. ペインタというモジュール化された図形構成手続きを利用して, 対応する図形を目で見, 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. 図形言語の拡張は既存のものとの統合性が取れるよう注意し, モデリングとレンダリングを内包するペインタ構成機能, さらに内部表現から外部スクリプトへの変換の機能を実装した. 実現した 3D 図形言語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り, その有効性を確認した.

謝辞

本研究の一部は科研費 24220006 の補助を受けた. また本稿中の図 15,16 として紹介した 3D 図形は受講生の枡井啓貴君による作品である.

参考文献

[1] 3D Systems, I.: Stereolithography Interface Specification, Technical report, 3D Systems, Inc., Valencia, CA, USA, June 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*, The MIT press, Cambridge, MA, USA, second edition, July 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] Dybvig, R. K.: *The Scheme Programming Language*, The MIT Press, Cambridge, MA, USA, fourth edition, July 2009.

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

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

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

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

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

[11] Inc., S.: Schemers Inc., July 2009. `http://www.eimacs.com/schemers.htm`.

[12] Kintel, M. and Wolf, C.: OpenSCAD - The Programmers Solid 3D CAD Modeller, July 2014. `http://www.openscad.org/`.

[13] 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.

[14] Phong, B. T.: Illumination for computer generated pictures, *Communications of the ACM*, Vol. 18, No. 6(1975), pp. 311–317.

[15] Reynolds, C. W.: Computer Animation with Scripts and Actors, *SIGGRAPH Comput. Graph.*, Vol. 16, No. 3(1982), pp. 289–296.

[16] 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 Bull.*, Vol. 25, No. 1(1993), pp. 288.

[17] Shiro, K.: Gauche - A Scheme Implementation, July 2014. `http://practical-scheme.net/gauche/`.

[18] Sierpiski, W.: Sur une courbe dont tout point est un point de ramification, *Compte Rendus hebdomadaires des sance de l'Acadmie des Science de Paris*, Vol. 160, No. 1(1915), pp. 302–305.

[19] Springer, G. and Friedman, D. P.: *Scheme and the Art of Programming*, The MIT Press, Cambridge, MA, USA, October 1989.

[20] Stratasys Ltd.: uPrint SE 3D Printer Pack for 3D Modeling — Stratasys, July 2014. `http://www.stratasys.com/3d-printers/idea-series/uprint-se`.

[21] 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.

[22] 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`.

[23] 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.

[24] 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.

[25] 奥乃博, 馬谷誠二, 糸山克寿: Syllabus - 京都大学 工学部, April 2013. `http://www.t.kyoto-u.ac.jp/syllabus-s/?mode=subject&lang=ja&year=2013&b=6&c=91150`.