

設計業務への Script-based CAD 導入の試み

株式会社 AIS 北海道 ○佐藤 佳美, 藤浦 巖, 高嶋 英敏

要旨

設計における CAD システムは、GUI 及び種々のコマンドをキーボード・マウス操作しながら入力する複雑な操作を習得する必要がある。さらに、その操作手順を記録し再現することは簡単ではない。一方、Script-based CAD はモデル作成手順を Script 言語で表現し、作成手順の記録、再現を容易にし、高効率な設計作業を可能にする。本研究では Script-based CAD の特徴と利点を明らかにし、設計業務への適用の可能性を調査することを目的とする。

1. はじめに

機械設計とは有限な形状表現要素(円・直線・曲線等)を用いて対象の形状および要件を表現するプロセスである。CAD システムはそのプロセスを実現するツールであり、今日までの普及は、形状表現要素の数学的表現に関する理論研究や、それらの要素の組み合わせで物体の表面形状を表現する技術の進展、その標準形(例えば STEP)が整備されることによって実現された。近年、新しい CAD システムとして、Open Source Hardware の考え方にに基づき Script 言語で形状表現する Script-based CAD の開発が進んでいる[1]。本報告では Script-based CAD と従来 CAD システムの比較、及び Script-based CAD の形状作成能力についての調査結果を報告する。

2. Script-based CAD と従来 CAD システムの比較

今回の比較調査では、Script-based CAD 及び従来 CAD システムを用いて図 1 の形状を作成し、保存形式、設計データの取り扱いについて考察を行う。ここで、Script-based CAD として CadQuery、従来 CAD システムとして FreeCAD を選択した。これらは、Free Open Source Software の CAD システムを調査した Felipe Machado らの文献[1]に基づく表 1 の CAD システムを候補とし、Viewer 表示ができること、STEP ファイルの入出力などデータ交換規格に対応していることを評価基準として選定している。従来 CAD システムは、GUI による形状作成機能とマクロコマンドによる形状作成機能を持っていることを踏まえ、①従来 CAD システムの GUI による形状表現方法(図 2)、②同マクロコマンドによる形状表現方法(図 3)、③Script-based CAD による形状表現方法(図 4)について調査した。

まず、保存形式に着目して考察する。FreeCAD では、一般的な従来 CAD システムと同様に Binary 形式、STEP 形式に加え、マクロコマンド(図 3)として保存することが可能である。CadQuery は Script 言語(図 4)と STEP 形式による保存が可能である。いずれも STEP 形式で保存可能であるが、この形式は異なるシステム間のデータ交換を目的としているため、記録された形状の可読性が低く、編集には GUI をベースとしたソフトウェアが必要である。一方、マクロコマンドや Script 言語は、人が編集することを前提としているため、記録された形状の可読性が高く、また、ソフトウェアを使用せずに編集することができ、さらに、図 3、図 4 に示すように設計意図の記録が容易である。ここで、

表 1 Free Open Source Software の CAD システムの例

CAD システム	表現方法	Viewer 表示	STEP 出力
OpenSCAD	Script 言語	可	不可
FreeCAD v0.19	従来 CAD システム マクロコマンド	可	可
CadQuery v2.0	Script 言語	可	可

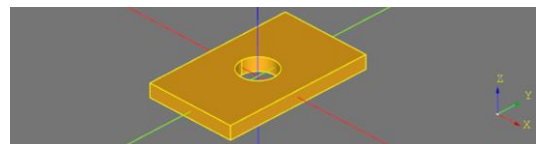


図 1 比較対象とする形状

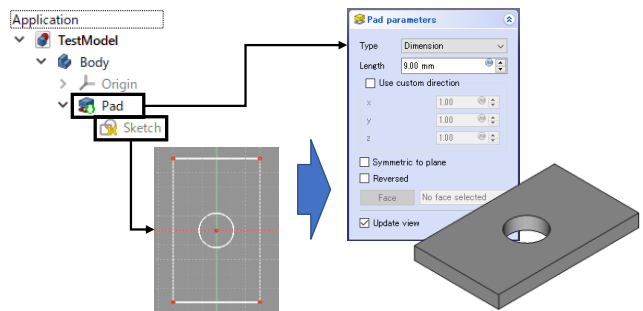


図 2 ①従来 CAD システムによる形状表現方法

```
# 空のBodyを作成し、空のスケッチを作成する(コマンド説明)
App.getDocument('TestModel').addObject('PartDesign::Body', 'Body')
App.getDocument('TestModel').getObject('Body')
    .newObject('Sketcher::SketchObject', 'Sketch')
App.getDocument('TestModel').getObject('Sketch').Support =
    (App.getDocument('TestModel').getObject('XY_Plane'), [''])
App.getDocument('TestModel').getObject('Sketch').MapMode = 'FlatFace'

# 長方形の板を定義(設計意図)
# スケッチにラインを作成(コマンド説明)
geoList = []
geoList.append(Part.LineSegment(App.Vector(-30.0, 50.0, 0.0), App.Vector(30.0, 50.0, 0.0)))
geoList.append(Part.LineSegment(App.Vector(30.0, 50.0, 0.0), App.Vector(30.0, -50.0, 0.0)))
geoList.append(Part.LineSegment(App.Vector(30.0, -50.0, 0.0), App.Vector(-30.0, -50.0, 0.0)))
geoList.append(Part.LineSegment(App.Vector(-30.0, -50.0, 0.0), App.Vector(-30.0, 50.0, 0.0)))
App.getDocument('TestModel').getObject('Sketch').addGeometry(geoList, False)

# 板の中央に穴をあける(設計意図)
App.getDocument('TestModel').getObject('Sketch').addGeometry(Part
    .Circle(App.Vector(0.0, 0.0, 0.0), App.Vector(0.0, 0.1, 12.0), False)

# スケッチからPadを作成(コマンド説明)
App.getDocument('TestModel').getObject('Body').newObject('PartDesign::Pad', 'Pad')
App.getDocument('TestModel').getObject('Pad').Profile = App.getDocument('TestModel')
    .getObject('Sketch')
App.getDocument('TestModel').getObject('Pad').Length = 9.0
App.getDocument('TestModel').getObject('Pad').Direction = (1, 1, 1)
App.getDocument('TestModel').getObject('Sketch').Visibility = False
App.ActiveDocument.recompute()
```

図 3 ②マクロコマンドによる形状表現方法

```
# 板の中央に穴をあける(設計意図)
result = (cq.Workplane('XY').box(60.0, 100.0, 9.0)
    .faces('>Z').workplane().hole(24.0))
```

図 4 ③Script-based CAD による形状表現方法

設計意図を含む形状の記録は、設計者間で生じる誤認識の防止や、設計ノウハウの再現に有効であり、その標準化の必要性が指摘されている[2]。したがって、マクロコマンドや Script 言語はその実現に適しているといえる。以上のことから、マクロコマンド、Script-based CAD による形状表現方法は、設計データの可読性を向上し、CAD ソフトウェアを使用しないオフライン設計を可能にすることに加え、設計意図の記録にも適していると考えられる。

次に、設計データの取り扱いに着目して考察する。設計変数を変更する際、従来 CAD システムでは CAD ソフトウェアの起動、形状や GUI に対する操作等が必要となり、作業工数が多くなる (図 2)。それに対して、図 3 のマクロコマンドでは、前述のようにソフトウェアの起動や GUI 操作を行わずに、設計変数を直接テキストで編集できるため、パラメトリックな形状変更が容易である。しかし記述内容は、GUI で操作する手順を細かく分解しマクロコマンドに置き換えて記録するため、長文且つ煩雑で直感的な理解が困難である。この課題に対し、Script-based CAD では、対象の形状を形状表現要素やプリミティブな形状のみで直接的、且つシンプルに形状を表現出来る (図 4)。以上のことから、①～③の中で Script-based CAD による形状表現方法が最も可読性が高く、パラメトリックな形状変更が容易であり、効率的な設計手法を実現できる可能性が高いと考えられる。

3. Script-based CAD の形状作成能力に関する調査

実際の設計業務で要求される複雑な形状の作成に対して、Script-based CAD がどの程度適用できるか、3つの形状を用いて確認した (図 5～7)。まず、曲面を持つ単一形状の作成可否を確認するために、スプライン形状をベースとした押し出し形状 (図 5) を作成した。結果、スプラインの制御点に対し、編集し易く整理された表現となった。次に単一形状を組み合わせた形状の作成可否を確認するために、Heatsink (図 6) を作成した。結果、2つの関数に変数を入力するのみの短い Script 言語で表現することができた。また、同一の単一形状であればパターン配置することにより、効率的に様々な寸法と配置の組み合わせを検討できることがわかった。次に複数の異なる断面を結ぶ形状に対する作成可否の確認として Wind Turbine を作成した (図 7)。Wind Turbine での設計では、構造・空力・システム・コントロールなどに対する様々な検討を行う必要があり、パラメトリックに設計データを扱うことが求められている[3]。作成の結果、複数の翼断面をつなげることが出来たが、一部で歪んだ面が生成された (図 7)。複数の面を滑らかにつなぐにあたり、CadQuery 内の数学的表現において、面の数と同じ次数の関数が使われていると仮定し、回避策を模索しているが、解決に至っていない。

これらのことより、Script-based CAD は、曲面を持つ形状、及び組み合わせによる形状について、作成可能なことがわかった。一方で、三次元的に断面がつながる複雑な形状の作成については、課題があることがわかった。

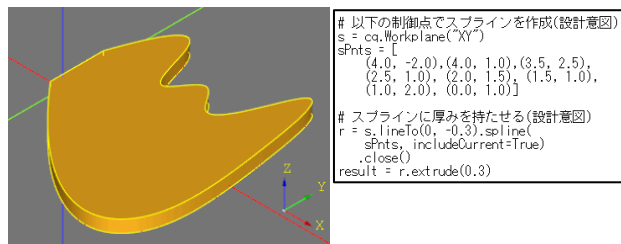


図 5 スプラインを用いた単一形状

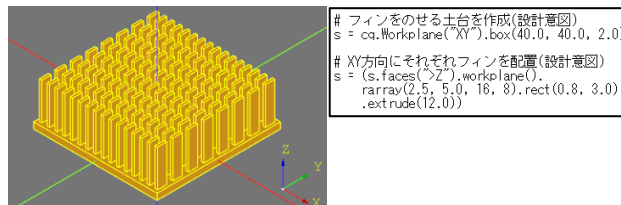


図 6 Heatsink

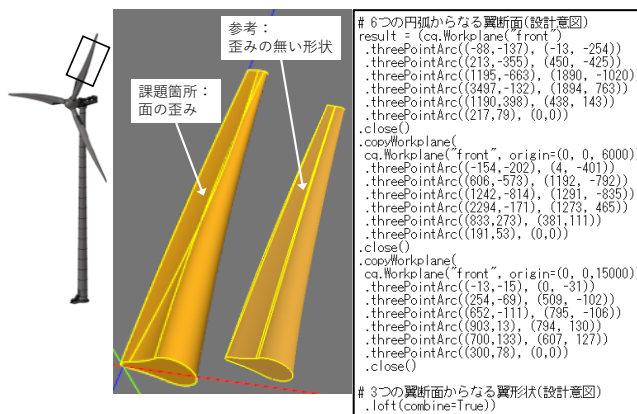


図 7 Wind Turbine

4. まとめ

新しい CAD システムとして開発されている Script-based CAD について、従来 CAD システムとの比較調査、及び形状作成能力の調査を行った。結果、Script-based CAD で保存された設計データは、可読性が高く、また、パラメトリックな形状変更が容易であり、さらに、ソフトウェアに依存せずオフラインで設計データを作成できるため、従来 CAD システムと比較して、効率的な設計手法を実現できる可能性が高いことがわかった。ただし、複雑な形状の作成には課題があり、作成手法の構築、又はツールとしての機能向上が必要と考えられるため、継続調査を行う。

参考文献

- [1] Felipe Machado, Norberto Malpica, Susana Borrromeo, Parametric CAD modeling for open source scientific hardware: Comparing OpenSCAD and FreeCAD Python scripts, pp.1-5, (2019)
- [2] Michael J. Pratt, Junhwan Kim, Experience in the Exchange of Procedural Shape Models using ISO 10303(STEP), pp.1-8, (2006)
- [3] Carlo L. Bottasso, Wind Turbine Design Origins of Systems Engineering and MDAO for Wind Energy Applications, pp.1-24, (2019)