DesignScript: origins, explanation, illustration

Robert Aish

"A programming language that doesn't change the way you think is not worth learning"

—Alan Perlis, 'Epigrams in Programming'

Abstract DesignScript, as the name suggests, is positioned at the intersection of design and programming. DesignScript can be viewed as part of the continuing tradition of the development of parametric and associative modeling tools for advanced architectural design and building engineering. Much of the thought processes that contribute to the effective use of DesignScript builds on the tradition of parametric design and associative modeling that is already widely distributed amongst the creative members of the architectural and engineering communities. Many of the existing parametric and associative modelling tools also support conventional scripting via connections to existing programming languages. The originality of DesignScript is that associative and parametric modeling is integrated with conventional scripting. Indeed, the definition of the associative and parametric model is recorded directly in DesignScript. But it is not what DesignScript *does* which is important, more what a designer can *do* with DesignScript. It is this change in the way you think that makes DesignScript worth learning.

1 Introduction

DesignScript is intended to be:

- a production modeling tool: to provide an efficient way for pragmatic designers to generate and evaluate complex geometric design models
- a fully-fledged programming language: as expected by expert programmers.
- a pedagogic tool: to help pragmatic design professions make the transition to competent programmer by the progressive acquisition of programming concepts and practice applied to design.

R. Aish
Director of Software Development, Autodesk

Essentially there are three themes interwoven here:

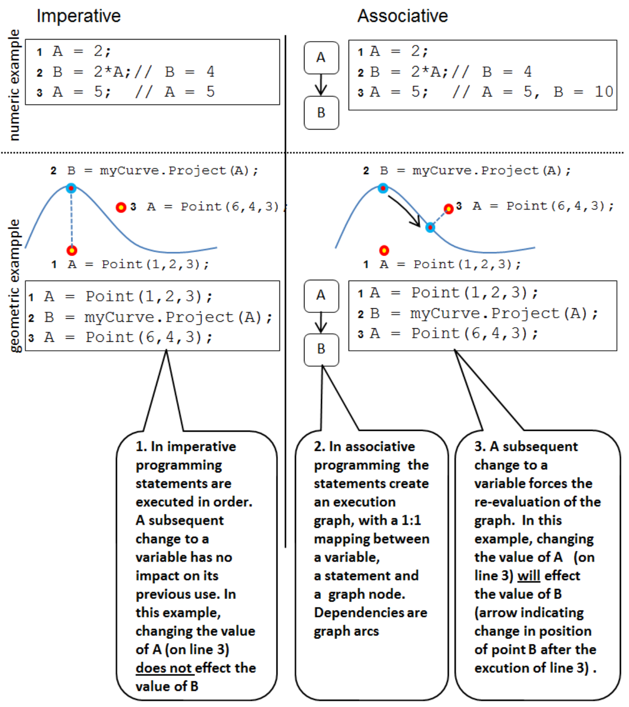
- The programming language theme: DesignScript as a programming language
- The design process theme: The use of DesignScript as a design toolset
- The pedagogic theme: using DesignScript as a way of learning how to design and to program.

2 Programming Language

From the perspective of a programming language, we might describe DesignScript as an *associative* language, which maintains a graph of dependencies between variables. In DesignScript these variables can represent numeric values or geometric entities, or other application constructs, including those defined by the user. The execution of a DesignScript program is effectively a *change-propagation* mechanism using this graph of variables. This change-propagation also functions as the update mechanism similar to that found in a conventional CAD application. However, unlike other CAD update mechanisms or associative and parametric modeling systems, in DesignScript this mechanism is exposed to the user and is completely programmable. Figure 1 illustrates the important differences between a conventional *imperative* language and an *associative* language such as DesignScript, while Figure 2 shows how a program statement in DesignScript can also be interpreted as natural language. Each term in the statement has an equivalent natural language interpretation so that whole statement can be understood by its natural language equivalent.

So a concise but somewhat complex description of DesignScript might be as a domain-specific, end-user, multi-paradigm, host-independent, extensible programming language (Fig. 3), as follows:

1. **domain-specific** DesignScript is intended to support the generation of geometric design models and therefore provides special constructs to assist in the representation of geometric models. More generally: A domain specific language may remove certain general purpose functionality and instead adds domain specific functionality as first class features of the language.
2. **end-user** DesignScript is intended to be used by experienced designers with a wide range of programming skill, ranging from non-programmers (who might indirectly *program* via interactive direct manipulation), to novice non-professional (end-user) programmers, and to experienced designers who have substantial expertise in programing. More generally: An end user language adds simplifying syntax to the language, while reducing some of restriction often associated with general purpose languages (intended for experienced programmers).
3. **Multi-paradigm** DesignScript integrates a number of different programming paradigms into a single language (including object-oriented, functional and associative paradigms) and introduces some additional programming concepts



```
1 A = 2;
2 B = 2*A;// B = 4
3 A = 5;   // A = 5
```

```
1 A = 2;
2 B = 2*A;// B = 4
3 A = 5;   // A = 5, B = 10
```

```
2 B = myCurve.Project(A);
3 A = Point(6,4,3);
1 A = Point(1,2,3);
```

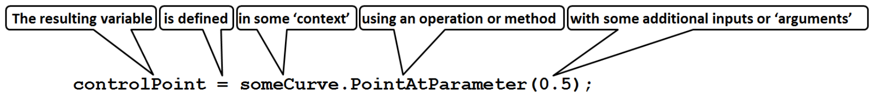
```
1 A = Point(1,2,3);
2 B = myCurve.Project(A);
3 A = Point(6,4,3);
```

1. In imperative programming statements are executed in order. A subsequent change to a variable has no impact on its previous use. In this example, changing the value of A (on line 3) <u>does not</u> effect the value of B

2. In associative programming the statements create an execution graph, with a 1:1 mapping between a variable, a statement and a graph node. Dependencies are graph arcs

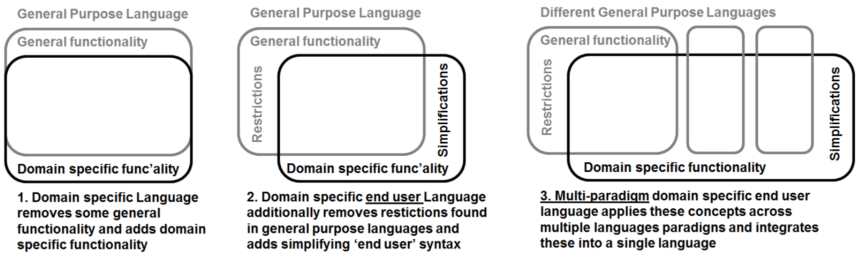
3. A subsequent change to a variable forces the re-evaluation of the graph. In this example, changing the value of A (on line 3) <u>will</u> effect the value of B (arrow indicating change in position of point B after the excution of line 3) .

Fig. 1 Comparing Imperative and Associative interpretation of the same program statements. It is this *change in the way you think* that makes DesignScript worth learning.


The resulting variable
is defined
in some 'context'
using an operation or method
with some additional inputs or 'arguments'

```
controlPoint = someCurve.PointAtParameter(0.5);
```

Fig. 2 Giving a natural language interpretation to a DesignScript statement.



1. Domain specific Language removes some general functionality and adds domain specific functionality

2. Domain specific <u>end user</u> Language additionally removes restictions found in general purpose languages and adds simplifying 'end user' syntax

<u>3. Multi-paradigm</u> domain specific end user language applies these concepts across multiple languages paradigns and integrates these into a single language

Fig. 3 How DesignScript differs from a regular general purpose programming language

that are relevant to the domain of generative design. More generally: A multi-paradigm language combines different programming styles into a single language and allows the user to select which paradigms or combination of paradigms are appropriate. (See Fig. 4)

4. **host-independent** DesignScript is intended to support the generation of geometric models and is therefore designed to be hosted within different CAD applications and access different geometric, engineering analysis and simulation libraries. For example, a DesignScript variable (based on specific class) may maintain a correspondence with a geometric entity in AutoCAD and simultaneously with entities within engineering analysis applications such as Ecotect and Robot.
5. **extensible** DesignScript can be extended by the user, by the addition of functions and classes.

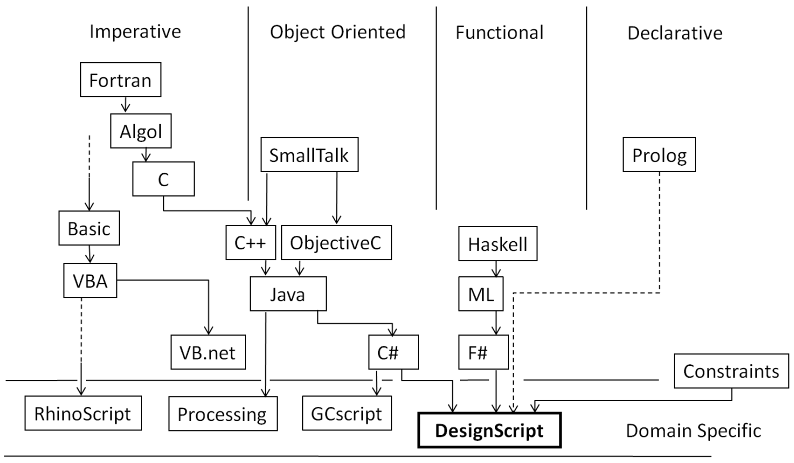


Fig. 4 The evolutionary *tree* for DesignScript (showing its precursors). DesignScript is a multi-paradigm language embracing imperative, objected oriented, functional and declarative programming concepts

3 Design Process

DesignScript is intended to support a computational approach to design which is accessible to designers who initially may be unfamiliar with this way of designing. Conventionally, computer-based design applications enabled the designer to create models which represent finished designs. The intention in developing DesignScript is to move beyond the representation of finished designs, and instead to support the designer to develop his own geometric and logical framework within which many different alternative design solutions can be easily generated and evaluated.

The development of DesignScript assumes that the designer wants to adopt this more exploratory approach to design and that he appreciates that this may involve some re-factoring of the design process so as to include a more explicit externalization of particular aspects of design thinking, for example:

- Explicitly identifying the key variables that drive the design
- Building the geometric and logical dependencies between these driver variables and the constructive geometry: potentially these dependencies can be complex *long chains*.
- Defining appropriate performance measures that can describe the resulting design solutions
- Exercising the complete model (by changing the design drivers and observing changes in the geometry and resulting performance measures) to explore more appropriate solutions
- Changing the geometric and logical dependencies in order to explore more alternatives

4 Pedagogic perspective

From a pedagogic perspective, DesignScript is designed around the concept of a learning curve and supports a very gradual approach to learning programming (Fig. 5):

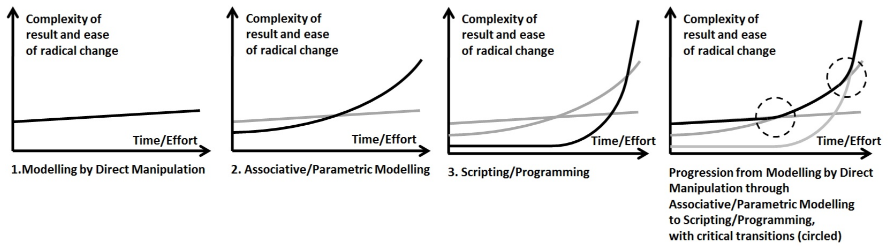


Fig. 5 DesignScript as conceived as a composite learning curve spanning different types of modelling and programming.

1. **For modelling by *direct manipulation***, the designer immediately obtains some interesting result for the modelling effort he makes, yet to change or refine or increase the complexity of the model may require an exhaustive amount of additional effort. Therefore the perceptive designer may search for a way to overcome the limitations of direct manipulation.

2. **For Associative or parametric modelling**, the designer may have to initially make some more effort to create the first associative model (than he did with regular modelling). Although the initial results may be unimpressive, he is investing in an associative model with higher semantic value. Because of this investment in design logic the designers ability to change and refine that model becomes comparatively easy (compared to non-associative modelling). The designer is not just investing his time and effort, but also has to learn new skills: in particular how to think associatively. However, the perceptive designer may recognise that some types of design logic are difficult to express in an associative modelling system, therefore the perceptive designer may search for a way to overcome the limitations of associative modelling.
3. **With scripting and programming**, considerable time and effort may be expended apparently without much evidence of success. Nothing works until it all works, but then the complexity of the model and the ability to re-generate the model with radically different design logic appears more powerful than what can be achieved with associative modelling.

   We can summarise this as:

- *Learning by doing*, for example, by interactive modelling
- *Learning by observing the correspondence between the DesignScript notation and geometry*, for example, by comparing the geometric model with the graph based symbolic model and with the DesignScript notation displayed in the IDE)

The following example illustrates the use of DesignScript. The design problem is to model a wave roof, based on a complex wave formation. The first step is to recognise that we should not attempt to directly model the wave formations with regular modelling tools. Instead we should recall that most complex wave forms can be constructed as the aggregate effect of simpler waves combined with related harmonic waves. In this case, the geometry is constructed by using a series of low and high frequency sine waves running orthogonally in the X and Y direction (Fig 6). The amplitude and number of peaks in the waves are controlled by root parameters. The X, Y and Z coordinates of the 2D field of points is defined by combining these sine waves (Fig 7). The number of peaks can be varied (Fig 8). The X, Y and base Z coordinates of the points can be derived from points in the UV parametric space of a surface, thereby giving the effect that the wave geometry is draped (and offset) from an underlying surface (Fig 9). Finally, the control vertices of the underlying surface can be modified giving the effect that the underlying surface is controlling the wave roof (Fig 10).

This presents the exactly the combination of direct modelling, associative modelling and scripting suggested in the learning curve in Fig. 5. It is not just the model (or the computation of the model) which is spanning this different approaches. It is the thought processes of the designer which is combining these different ways of thinking.

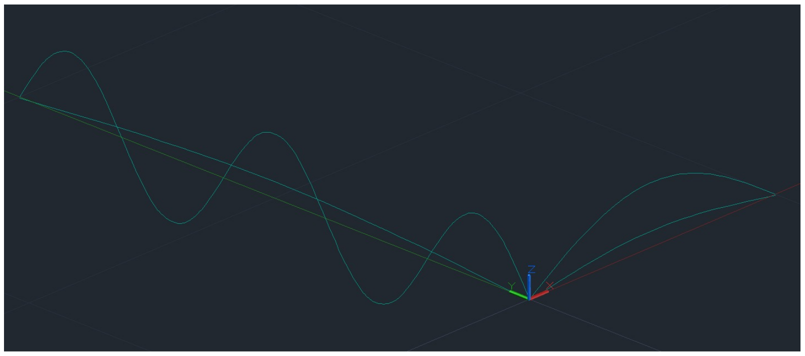


Fig. 6 High and Low frequency waves in the X and Y directions

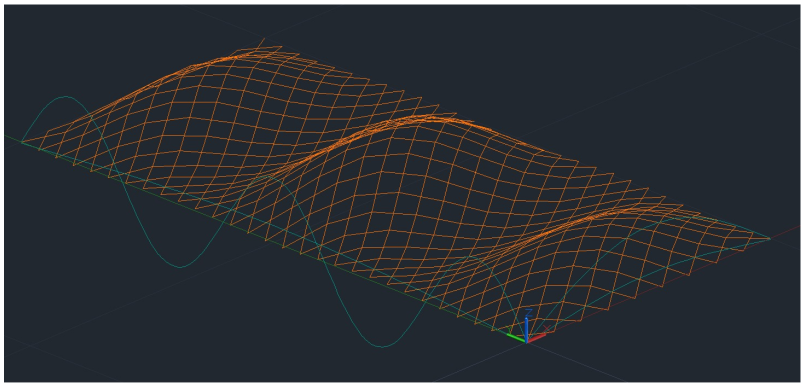


Fig. 7 The resulting wave roof is created by aggregating these orthogonal waves

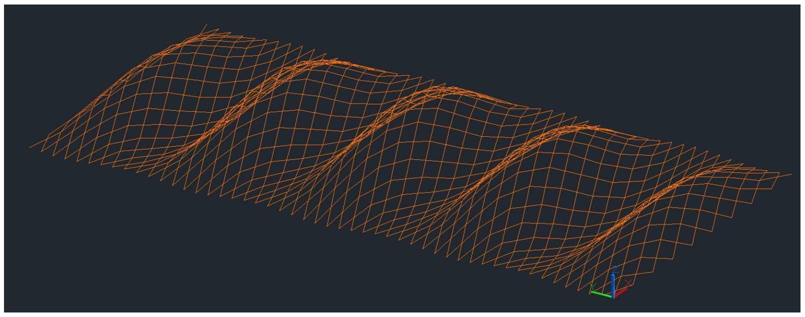


Fig. 8 The number of peaks can be varied

5 Discussion

The three themes which are interwoven here (the programming language theme, the design process theme and the pedagogic theme) all come together when we address

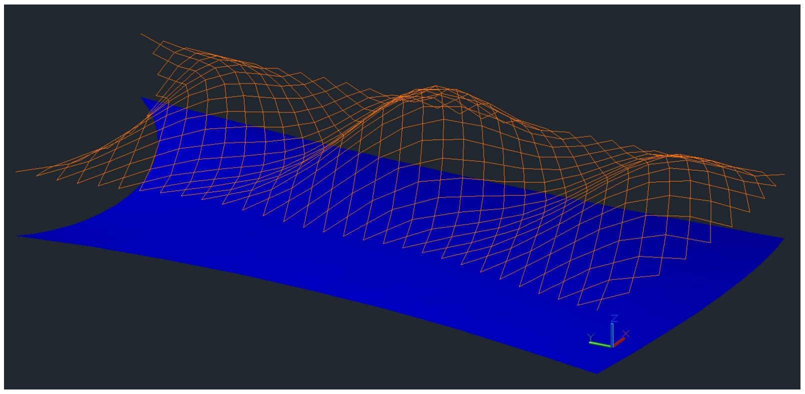


Fig. 9 Draping (and offsetting) the wave roof from an underlying surface

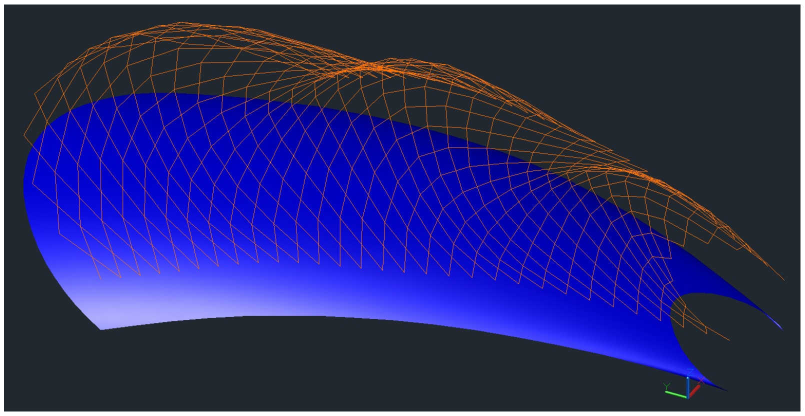


Fig. 10 The control vertices of the underlying surface can be modified giving the effect that the underlying surface is controlling the *wave* roof

the central issue: How can a computational tools invoke a computational mindset and in turn contribute to design thinking?

Using DesignScript is a new way of designing with its own expressive possibilities. But there is a level of understanding required to harness this expressiveness and this suggests a level of rigor and discipline. The argument is that the experience of learning and using DesignScript contributes not just to the expressiveness and clarity of the resulting design but also to the skills and knowledge of the user.

In short,“a new toolset suggests a new mindset”.