Butterfly CAD

Shareable & Editable 2D Parametric Specifications for Fabrication

Cheng Xu
Capital One Inc
San Francisco, CA 94110, USA
chengx.cn@gmail.com

Michael Philetus Weller
For Commas Inc
Seattle, WA 98104, USA
philetus@gmail.com

.

Abstract
We outline our plans for our Butterfly CAD project: a design tool for creating 2D parametric specifications that can be shared as a web page and edited through a browser. The goal of this project is to provide better communication of high-level design intent by allowing experts with domain knowledge (for example in areas such as fabrication or biomimicry) to embed their knowledge in designs as parameters and constraints; viewing these designs and adjusting the parameters will not require any specialized software beyond a browser.

Author Keywords
CAD; Parametric Design; Design Intent; CSCW.

ACM Classification Keywords
H.5.3. Group and Organization Interfaces: Computer-supported cooperative work.

Introduction
The advantage of high-level design tools is that they allow the authors of a design to express their idea at a level of abstraction above the particular placement of lines on the page. Parameters and constraints are capable of describing the invariant properties of a

system rather than the contingent features of one particular solution. [1]

However the tools that allow designs to be expressed at this level (to name a few examples: Catia, Solidworks and Inventor) in general have two significant barriers: a steep learning curve and a high sticker price. With the proliferation of inexpensive robotic fabrication tools it is often more expensive and difficult to run design software than to run a fabrication robot capable of actually manufacturing designs.

In this design ecology a designer may spend a great deal of effort creating a high-level definition of a system on high-end software; then to have it fabricated the designer exports a file in one of the common digital exchange formats (such as scalable vector graphics for 2D or stereolithography for 3D). These digital exchange formats in general capture only the most contingent aspects of a design. But these are the files that a person outside the original design domain can actually view and get fabricated, so these are the files that are posted to online repositories such as Thingiverse or passed between domain experts with different specializations.

One of the more subtle negative externalities of this ecosystem is that knowledge about particular domains that has been painstakingly digitally modeled is kept hidden from people in other domains. Our Butterfly CAD project has two goals:

1. to create a digital exchange format that encourages domain experts to share high-level models of their designs; and
2. to lower the barrier to interacting with parametric designs.

By implementing our system using web technologies we will make it possible to simply post a high-level design as a web page and allow people to directly tweak the parameters through a browser interface. By including a simple GUI API we can encourage designers to expose key parameters as slider bars or draggable elements. We envision people using our system to create interfaces in the vein of Nervous System's Radiolara app [2] which allows people to manipulate a bio-inspired mesh structure within their browser. Another button click (and a credit card number) sends the custom design to be fabricated.

Below we give a more detailed description of our implementation plan for Butterfly CAD.

System

One of the biggest challenges of this project is encouraging the widest possible group of people to be able to open and edit design documents. To this end we are building a web-based interface. Specifically we are implementing the system as a client-side Javascript application that renders through making WebGL calls. By distributing the application Javascript along with a design each document becomes a standalone web application that can be opened and run on any system with an HTML5 browser.

As Javascript is not our favorite language to develop in we are actually writing the system in Go and transpiling the code into Javascript with GopherJS [3]. The system has two main subcomponents: a lower-level 2D Bezier triangle mesh renderer and a higher-level parametric

description parser. The parser takes our parametric description and translates it into a Bezier triangle mesh that is then fed into our low-level renderer.

We have already implemented the low-level rendering layer. [4] It renders a 2D triangle mesh format inspired by the 3D triangle mesh of the stereolithography format. The first section of our "flyspek" mesh data structure features a list of vertices, the second a list of colors, and the third has a list of triangles defined in terms of the vertices and colors specified in the first two sections. Individual triangles can also be specified as filled, concave or convex. Concave and convex triangles are divided by a quadratic Bezier curve (with the 3rd point of the triangle controlling the degree of curvature) with only one side or the other shaded. Our lower level rendering system applies Loop and Blinn's technique [5] to leverage GLSL shaders to render antialiased quadratic Bezier curves using the system graphics hardware.

We are currently exploring the design of our higher-level language for describing 2D zones bounded by straight edges and Bezier curves. One of the main elements is an abstract container (called a "zib" after Xzibit, the internet's biggest fan of recursion) that can serve as an archetype for further instances and can be transformed by a matrix; by combining these two features arrays and grids of elements can be defined. Figure 1 shows an illustration demonstrating what a high-level description of the outline of a butterfly wing could look like. The basic structure is a 2D "zone" is defined by extent and gap "boundaries". The zone in our example has no gaps, so there is just one extent boundary given. A boundary is in turn defined by referencing a list of "guides". Each guide is defined in the context of a zib and is represented as two 2D vectors, one defining the offset from the current zib's anchor, and the other serving to define the curvature of the boundary as it passes through the guide. Each zib can be anchored by a transformation matrix. In the example the "leftWing" zib is anchored with a translation matrix warp and contains several guides. The "rightWing" zib is an instance of the left wing, except it overrides the anchor with a translation matrix that mirrors its guides across the spine to the right. The wing zone boundary is then generated by referencing these guides in the proper order.

References

1. Mark D Gross. 1986. *Design as Exploring Constraints*. Ph.D Dissertation. Massachusetts Institute of Technology (MIT), Cambridge, MA.
2. Nervous System. Radiolara: Bio-inspired Design App. 2007. Retrieved January 2016 from http://n-e-r-v-o-u-s.com/radiolaria/.
3. GopherJS: A Compiler from Go to Javascript. Retrieved January 2016 from https://github.com/gopherjs/gopherjs/.
4. Flyspek: A 2D form description library. Retrieved January 2016 from https://github.com/philetus/flyspek.
5. Charles Loop and Jim Blinn. 1995. Resolution independent curve rendering using programmable graphics hardware. In *SIGGRAPH*, 1000-1009.

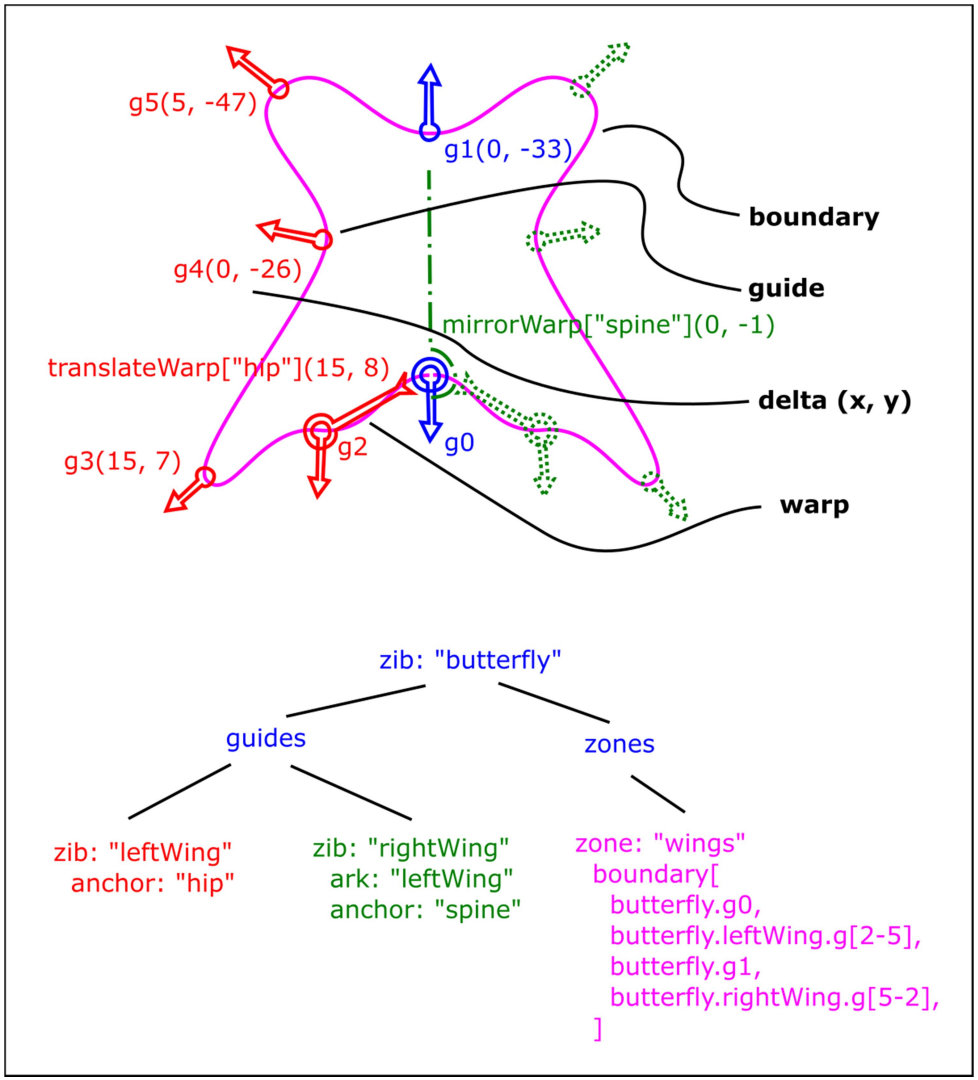

Figure 1: An illustration showing a speculative version of the data structure for a high-level Butterfly CAD specification.