Presented at the “Let's Get Physical” workshop at the 2nd International Conference on Design Computing and Cognition (DCC ’06), Eindhoven, Holland, July 8, 2006.

TRISKIT

A software-generated construction toy system

FRED MARTIN, MARTIN MEO, GEORGE DOYLE
University of Massachusetts Lowell, USA

Abstract. Triskit is a modular construction toy system designed to be outputted from acrylic sheet stock on small laser cutters. Even the best commercially construction toys (e.g., LEGO; K’NEX) are limited in that users have access to only the parts provided by the manufacturer. We wanted a construction system that comes with software for easily generating itself. This paper presents the design process that led to the Triskit system and initial results in using the materials with 10- and 11-year-old students.

1. Introduction

Play with any well-designed construction toy system—such as LEGO Technic, K’NEX, or Lincoln Logs—and you develop an understanding of what makes the toy a *system*. Parts fit together in sensible, predicable ways, there are unit dimensions and normative interconnect methods, and a coherent visual and tactile experience emerges. Over time, you may conceive of and then desire a part that does not exist in the building system. But you know that it logically could exist as a legitimate element. Of course, commercial vendors do not typically publish their systems’ design drawings, so it is hard to realize your component.

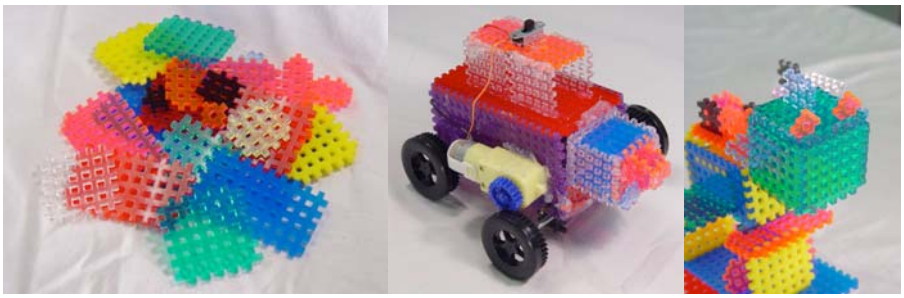

Figure 1: Triskit components; child’s motorized car; dinosaur sculpture

This limitation motivated the development of the Triskit construction toy system. We desired a set of components that we could easily manufacture, and for which we could provide the specifications to other end users, allowing them to create parts that would automatically become part of the Triskit system.

Figure 1 shows our results: the Triskit components, a motorized car built by 12-year-old, and a dinosaur sculpture designed by an undergraduate in our lab. The remainder of this paper describes our design process, our Java-based software tool for creating Triskits, and reflections after working with a group of 4th- and 5th-grade students.

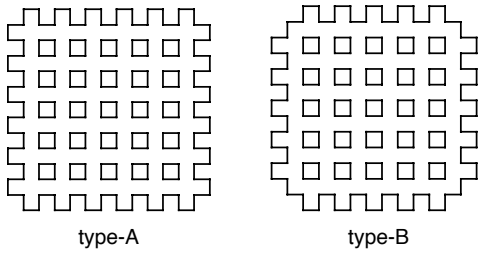

Figure 2. type-A and type-B Triskits.

2. Design Process

We selected 1/8" acrylic sheet stock as our base material, because it is relatively inexpensive, reasonably sound, and comes in pretty colors. Also, the laser cutter in our lab (a Trotec Speedy II with a 45W laser) can cut through it nicely.

As experienced LEGO builders, we took inspiration from this system. We wanted to be able to build models comparable in size and function to a typical motorized LEGO construction. We liked the fact that LEGO bricks are self-mounting; that is, a brick connects to another brick without the use of a specialized connector component.

In prior one-off projects we had done with our laser cutter, we used tab-and-slot designs to join parts. We generalized this approach and developed the first Triskit "waffle," which later became known as the "type-A Triskit" (Figure 2, left). (The Triskit name itself comes from our parts' resemblance to the popular snack cracker.)

When playing with our type-A Triskits, we quickly realized that the hole-slots on each Triskit face would be offset when two type-A Triskits are joined. This led to the design of the type-B Triskit, which has its edge fingers offset by a half-step (Figure 2, right). With type-A and -B

Triskits, we were able to build 4-sided rectangular structures with aligned holes.

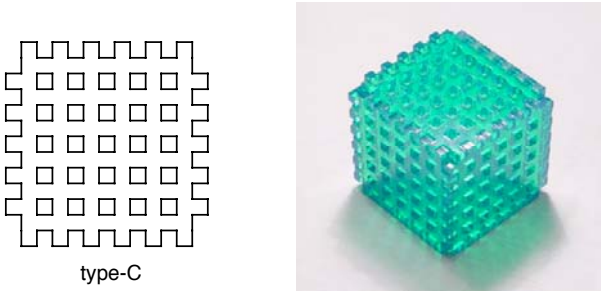

Figure 3: type-C Triskit and cube built from 6 identical type-Cs.

We then tried to close the 4-sided rectangles by putting Triskits on their two open faces, and realized that we could not. This led to the development of the type-C Triskit, which has type-A fingers on one pair of opposite sides and type-B figures on the other. We were now able to build the cube (Figure 3).

At this point, we felt like we had a design that was worth playing with, and we set off to develop software to easily generate variants of our 3 Triskit types.

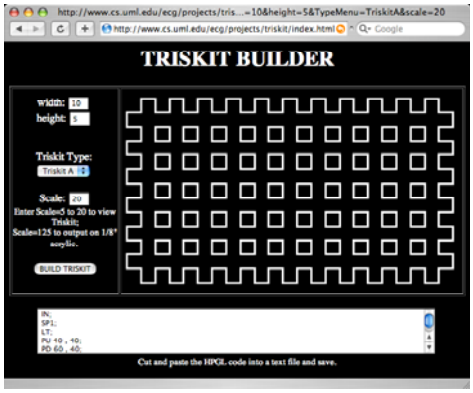

Figure 4: Triskit Builder Java applet.

3. Making Triskits

The first Triskits were drawn directly into a vector-based draw program (CorelDraw). After we had worked out the basic design and dimensional units, we built a prototype software tool in Microworlds Logo. As the Logo turtle nagivated the screen and drew the Triskit, its movements

were captured as HPGL[1] commands. The HPGL code was saved as a text file, which was then imported into a blank CorelDraw document and printed to the laser cutter.

We then developed a Java applet to do the same. After specifying the Triskit features (dimensions and type), the "Build Triskit" button displays the Triskit on the screen and generates the HPGL code for manufacturing it (Figure 4).

4. User Testing

We conducted a workshop with 4th and 5th grade students at a local school, meeting for an hour twice per week for a total of about 20 sessions. We introduced the Triskits at the beginning of the workshop and used them as the primary building material for the duration.

The children found the Triskits challenging; it was not evident to many of them how they should fit together. After there were a few models underway in the classroom, however, things got easier.

We realized that the Triskits do not hold together particularly well. This led to some quick improvisation. We tried screws and nylon cable ties to hold them together. 6×32 machine screws fit snugly in the hole-slots on each Triskit face, but the kids did not seem to like using them. Finally, we introduced hot glue guns, and things became a lot happier. A project model designed by one of the children, and representative of the group, is shown in Figure 1 (center photo).

One surprise was that several children did not constrain themselves to the right angle geometry inherent in the Triskit parts. They cut them, glued them at angles, and otherwise treated them as casually as necessary to obtain the effect they desired.

The Triskits do not include any parts to support motion (e.g. gears and axles). To allow the kids to motorize their models, we purchased generic plastic gears and wheels, metal axle rods, and gear motors.

We showed the children the Triskit Builder software on one occasion, but it did not seem terribly relevant to the children's goals (building a motorized model). In retrospect, the Triskit system differs fundamentally from other software-based building systems—e.g. Designosaurus (Oh *et al.* 2006) or HyperGami (Eisenberg *et al.* 2003). In these systems, the primary development is done on the screen; with Triskit, the main work is with your hands.

---

[1] Hewlett Packard Graphics Language, a vector-based drawing language originally designed for HP pen plotters. See http://en.wikipedia.org/wiki/HPGL.

5. Concluding Remarks

At present, the Triskit system is more satisfying for creating static, sculptural models (e.g., the dinosaur of Figure 1) than structurally sound models with moving mechanisms. After having designed the Triskits, we have an even greater appreciation for the power and sophistication of the LEGO Technic system.

One of the biggest challenges is getting the tab-and-slot connectors to work reliably. This is compounded by our discovery that 1/8" thick acrylic plastic varies significantly in its nominal dimension. Using 6×32 screws could work, but the children we tested with were not keen on that approach. We might have better luck with 1/4" plastic, or a more significant re-thinking of our approach.

We underestimated the time it takes for the laser cutter to produce a Triskit. The internal hole-slots create is a lot of cutting per piece.

The Triskits are presently limited to rectilinear forms, but as noted a number of kids found ways around these restrictions anyway. Creating diagonal shapes (e.g. triangles) would be relatively straightforward and could add a lot to the system, even in its primitive state.

The Triskit Builder applet works nicely, and it is quite easy to generate Triskits of varying dimensions. We would like to better integrate the applet with the overall software toolchain.

Looking ahead, we realize that we don't have to do everything with our laser. By combining Triskits or their descendants with inexpensive, off-the-shelf parts (e.g., screws, common plastic gears), we could have a building system that is powerful and flexible.

Acknowledgments

Martin Meo led the design of the Triskit system. Andrew Chanler solved the cube problem by inventing the type-C Triskit. George Doyle implemented the Java applet version of the Triskit Builder. David Ceddia co-taught the robotics workshop along with co-authors Fred Martin and George Doyle. Michael Piantedosi created the Triskit dinosaur shown in Figure 1, and photographed all of the models shown in this paper.

References

Eisenberg, M, Eisenberg, A, Hendrix, S, Blauvelt, G, Butter, D, Garcia, J, Lewis, R, and Nielsen, T: 2003, As We May Print: New Directions in Output Devices and Computational Crafts for Children, In *Proceedings of Interaction Design and Children*, Preston, UK, pp. 31–39.

Oh, Y, Johnson, G, Gross, MD, and Do, E: 2006, The Designosaur and the Furniture Factory: simple software for fast fabrication, *International Conference on Design Computing and Cognition.*