Enclosed: A Component-Centric Interface for Designing Prototype Enclosures

Christian Weichel
Lancaster University
c.weichel@lancaster.ac.uk

Manfred Lau
Lancaster University
m.lau@lancaster.ac.uk

Hans Gellersen
Lancaster University
hwg@comp.lancs.ac.uk

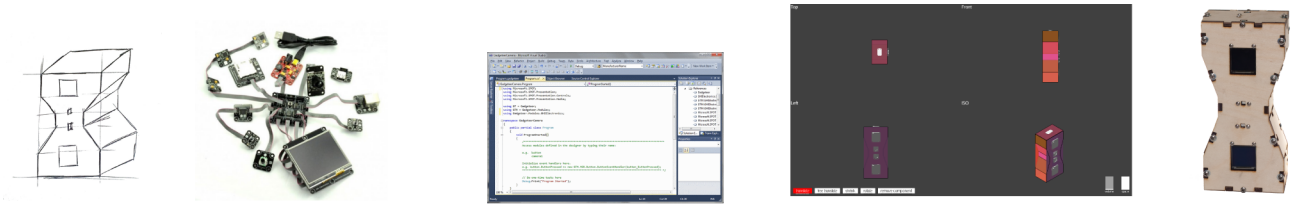

Figure 1. The design process for a device prototype: starting from an idea, hardware components are selected (and with them a prototyping framework - e.g. .NET Gadgeteer, Arduino or generic components). Once the components are connected together, software can be written for the device (the Gadgeteer platform already provides this). This paper focuses on a tool for virtually designing an enclosure, which can then be fabricated into a physical prototype.

ABSTRACT

This paper explores the problem of designing enclosures (or physical cases) that are needed for prototyping electronic devices. We present a novel interface that uses electronic components as handles for designing the 3D shape of the enclosure. We use the .NET Gadgeteer platform as a case study of this problem, and implemented a proof-of-concept system for designing enclosures for Gadgeteer components. We show examples of enclosures designed and fabricated with our system.

Author Keywords

rapid prototyping, personal fabrication, 3D modeling, user interfaces, user-generated design

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation: User Interfaces

INTRODUCTION

The process of prototyping a new device not only includes connecting its components and writing its software, but also includes designing and building an enclosure for the device. Designing an enclosure is an important step, as the aesthetics and haptics of the device are determined by how it is shaped and what material is used. Traditional computer-aided design (CAD) tools are complex and require extensive training. Designing an enclosure with such tools is cumbersome, as each part of the enclosure has to be designed individually and so that it fits with the others, including the joints/screws connecting the panels.

This paper explores the problem of designing and fabricating enclosures for electronic devices, and we build a tool to ease the prototype process of a physical enclosure. Our interface is novel in that it focuses on *using the electronic components as handles for designing the shape of the enclosure*, in contrast to typical CAD modeling tools that manipulate edges and vertices to modify a 3D shape. Creating a design from scratch can also be a daunting task. By creating an enclosure using the information about the components that need to be placed, we free the user from this "blank canvas syndrome". The design can then be iteratively refined using the components as handles. By focusing on laser cutting as the production technology, we constrain the possible operations so that the outcome is always realizable on a laser cutter, and further reduce the interface complexity. The user does not have to care about the laser cutter requirements, specific joint techniques, or the component requirements for holes/mounting points.

We use the existing .NET Gadgeteer platform [11] as a case study. Our implementation is a proof-of-concept system that extends Gadgeteer to fabricate enclosures with a laser cutter. Gadgeteer provides a rapid platform for designing, prototyping and fabricating electronic devices with a set of smaller electronic components. Fully functional devices can be prototyped, built, and programmed within a number of hours. However, the design of the physical enclosure can be time-consuming as modeling a 3D shape is a difficult problem. Creating a 3D model that fits with existing electronic components is even more difficult. The current approach utlizes a

Figure 2. Three alarm clocks, all of which share a similar bill of components but exhibit different shapes and behavior. On the left side is the *brick* prototype which represents the simplest alarm clock. In the center is the *pool* enclosure which has the buttons on the angled side panels. The third prototype on the right-hand side, bears deliberate resemblance with an *hourglas* and can be snoozed by turning it arround.

proprietary 3D modeling software, and the process typically requires a few hours. On the other hand, our system can be used to design an enclosure in the order of minutes.

Related Work

With the emergence of personal fabrication [6, 8], easy-to-use tools for designing customised objects are becoming common. Previous work that is closely related to our work includes the design and fabrication of storage boxes (for storing objects inside them) [2, 9]. Sketch-based interfaces are often used for creating 3D shapes for graphics and fabrication purposes [7, 5, 10]. Other tools used include TinkerCAD and SketchUp [3, 1], both of which lack specific production technique support. Our work focuses on an easy-to-use interface for creating enclosures using a laser cutter, that fit with existing electronic components.

USER EXPERIENCE AND INTERFACE

We describe the user experience by explaining the design process (Figure 1): from having an initial idea to fabricating a physical prototype. Suppose we wanted to create prototypes for three alarm clock implementations (Figure 2). The first one: *brick*, is a conventional design using a display and two buttons at the top with which the user can set the time and snooze the alarm. *Pool* works in a similar way, but has the two buttons on angled panels on the side, possibly making the alarm clock more economic. Our third alarm clock prototype bears deliberate ressemblance with an *hourglass*. Time is set using a button on either side and the clock is snoozed by turning the hourglass around.

Connecting components and writing software

The design process starts with an idea of what the user wants to build. Based on that understanding, the user selects a set of components. For the alarm clocks, we select displays, buttons, LEDs, batteries, accelerometers and a main board. Gadgeteer [11] provides a visual hardware designer for connecting the components together, and a development environment for writing the software to control the components.

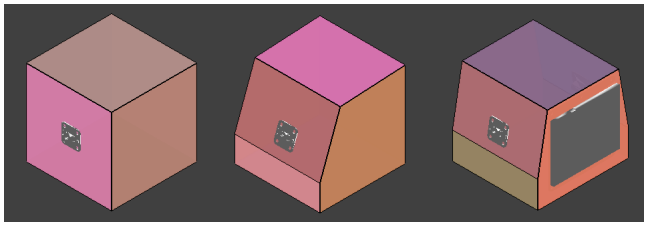

Figure 3. Left: The initial 3D shape is a box, and one can translate components to resize enclosure panels. Middle: Users can rotate components to rotate/split/join enclosure panels. Right: Our *pool* alarm clock design.

Starting the enclosure editor

Once the hardware components have been connected and the software is written, we start designing the enclosure. The editor receives the bill of components from the Gadgeteer development environment and starts by presenting a list of the components. Components can either be internal or external. Typical examples of internal components include batteries and WiFi/Bluetooth modules. External components serve their main purpose by explicitly interacting with the user. Displays, buttons, LEDs, wired connections (for ethernet and power), and actuators are examples of such modules. Confirming the list of components leads to our enclosure design tool (Figure 1 enclosure design). We are presented with four views of the enclosure (top/bottom, front/back, left/right, ISO), an action indicator bar at the bottom (for making selections on how to translate and rotate components on the associated panels), and the component palette on the right. The editor provides a box as the initial shape which the user then *refines by translating and rotating components on the panels (or faces) of the enclosure* (Figure 3).

Example: buzzing inside or outside

For our alarm clock examples, we have at least one component whichs role is not clear: the buzzer is an internal component in that it is not operated by the user or would require user visibility. However, a buzzer is likely to be louder when the sound produced by it doesn't have to penetrate enclosing walls (if the buzzer is external). The user can choose whether a buzzer is internal or external.

Translating components to resize enclosure panels

Components can be added to any face of the enclosure by dragging them from the palette to the desired face. If there is no space on the target face, or the component is too big to fit on the face, the enclosure will resize automatically to provide space for the new component. Once placed on a panel, components are represented by their 3D model provided by the Gadgeteer platform (Figure 4 shows an example 3D model). Moving the component on the panel not only translates the component, but also resizes the panel. There are three possibilities depending on the currently active action:

- *Translate* action: the component is translated on the plane of the panel only and never away from it. The enclosure does not shrink when the component is moved away from the panel but the component "snaps" back to it.
- *Free Translate* action: the component can be moved away from the plane of the panel, causing the enclosure to resize

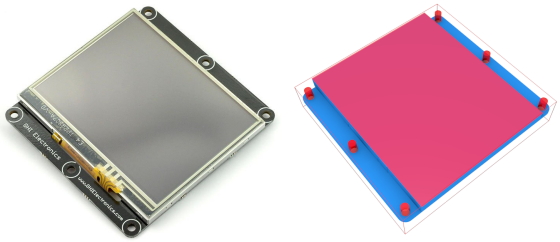

Figure 4. (Left) The Gadgeteer T35 display component manufactured by GHI electronics. (Right) The 3D model (in blue) as provided by the Gadgeteer platform, and the additional annotations (in red, mounting holes/cut-outs and bounding box) to use it with our editor.

(grow and shrink) to again enclose the component.

- *Shrink* action: shrinks the enclosure when the component is moved towards the inside of a panel.

The user can translate the components in a continuous mode (default) or discrete mode (snap to nearest 5mm). Components can also be removed from a panel and placed back in the palette.

Example: choosing the panels

The *brick* example has components placed on two different panels: the display on the front panel and two buttons on the top one. Such a configuration is quickly created by dragging the components from the palette to their desired place, resizing the panel in the process.

Rotating components to rotate/split/join enclosure panels

Clicking on and rotating a component automatically rotates the panel that the component is connected to (Figure 3 middle). The edge around which the panel is rotated is determined by which edge intersects the selected component's *manipulation range* (a fixed margin around the component). The edge of rotation is chosen by checking which horizontal edge intersects with the manipulation range. If multiple horizontal edges intersect in the manipulation range, the lowest edge (the edge with the smallest y-coordinate) is chosen. We limit rotation to horizontal edges, as practice has shown that rotation around vertical edges is seldom needed and often leads to enclosures unproducible with laser-cutting. If no horizontal edge intersects with the manipulation range, a new edge is created at the lower horizontal end of the manipulation range, and splitting the existing panel into two panels. When the angle between two panels is reduced to a value less than 2.5 degrees, the editor joins the two panels to become one and removes the edge that previously separated both. The user can rotate the components in a continuous mode (default) or discrete mode (snap to nearest 10°).

Example: slanting and the hourglass

For our *hourglass* example split the panels and rotated them to achieve the desired shape by placing a component on the bottom/side panels. We repeated the split/rotate step four times for each side, requiring a total of 8 steps to create the shape. The number of steps needed to achieve the same (including joints) in a traditional CAD system is about 10 times higher.

IMPLEMENTATION

We represent an enclosure as a graph loosely resembling its bill of material. An enclosure consists of a set of panels, components, and the connections between panels and components. Hence we represent an enclosure as a graph with panels and components represented by vertices and the edges representing the connections between the entities. A component consists its 3D model, bounding box and a set of holes representing the outlets a component requires on the enclosure. These holes are modeled as circles and rectangles placed on a face of the component's bounding box, with respect to the center of that face. We enumerate the bounding box faces to uniquely identify to which BB face a hole belongs to. Our model allows *panel to component*, *panel to panel* and *component to component* connections. A *panel to component* connection includes the position of the component (oriented at the center of the component's bounding box) on the panel.

Translating component on panel

To move a component on a panel, we first determine if a panel edge intersects with the component's padded bounding box (BB + padding). If such an intersection is found and the component is moved towards the outside of the panel, we move the other panel adjacent to the intersecting edge, so that the edge no longer intersects with the padded bounding box. In case the component is moved towards the inside of the panel and shrinking is allowed, we move the adjacent panel by the component's displacement vector, hence shrinking the enclosure. After all these operations, the component position is updated by storing the new position in the respective *panel to component* connection.

Rotating component on panel

Rotating a component on a panel can cause the panel to be subdivided. A subdivision edge must satisfy these criterion: (i) it must intersect with exactly two existing panel edges (can be violated by convexity); (ii) it must be parallel to an existing edge, so that the outcome of the subdivision is still fabricatable; and (iii) it must not intersect with a component on the panel. We always use the edge parallel to the horizontal panel axis going through the lowest point(s) of the component's manipulation range. To subdivide the panel, we create two panels to substitute the existing one. First we find the two edges intersected by the subdivision edge[1], which also determines which vertices belong to which panel. After creating the new panels, the connections of the old panel are mapped to the new panels. Each old *panel to panel* connection is replaced with at least one, and at maximum two new connections. All previous *panel to component* connections are mapped to either of the two new panels: we use a ray-casting algorithm to determine in which panel the component center resides.

FABRICATION AND RESULTS

Outline generation

The outline generation algorithm is an essential part of the system and serves the purposes of generating outlines to be

[1]If there were more than two intersecting edges, criteria (i) would be violated.

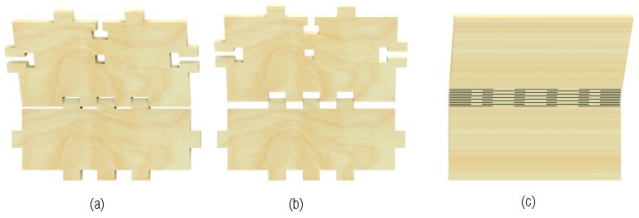

Figure 5. Three different joints at the same angle. (a) was produced using our $\sin$ rule. (b) was produced with the naive approach resulting in a too conservative size reduction. (c) is a possible way of "joining" panels that exploits the flexibility of the production material.

cut using a laser cutter or CNC router and checking for certain production specific issues in the model. Outlines are generated for each panel independently of the other panels. First, each panel edge is assigned a role, based on its role in the enclosure graph: if the current panel A assumes a source role in the connection to its adjacent panel B, the edge is marked male and female otherwise. Second, male edges are shrunk to account for material thickness. The amount an edge is reduced depends on the angle of inclination with the adjacent panel. We found that reducing an edge by $m \sin(A \angle B)$, where m is the material thickness and $A \angle B$ is the angle of inclinination between the two panels, produces aesthetically pleasing results (Figure 5a,b). Third, we add joints to connect the panels. This system is capable of adding three types of joints, depending on the available space: pure finger joints, screw joints and a combination of the two. Finger joints are always 10mm wide, and the largest odd number of finger joints that fits into the space is added to each edge. If an edge has at least three or more finger joints, the middle joint becomes a screw joint. If an edge has only one finger joint and there is enough space, the finger joint is replaced with a finger/screw joint combination (which is 15mm wide to account for the screw diameter). Joint gender is based on the role determination in step one.

Placing the outlines on sheet material

The 2D packing problem is a well studied one with many different problem instances. We consider placing the outlines on the sheet material as an instance of the strip packing problem: place a set of rectangles of different size on a strip of fixed width but infinite height, so that the total height is minimized. To compute the layout, we use the *First-Fit Decreasing Height* packing algorithm [4] as it has a low computational complexity $\mathcal{O}(n \cdot \log n)$. As the panels tend to be of similiar size, the algorithm produces good results.

Results of fabricated enclosures

Figure 2 shows the three physical enclosures that we built. All three enclosures were cut out of 600x300x6mm plywood sheets, and assembled using M3 screws for mounting the components and M4 screws for all screw joints.

DISCUSSION, LIMITATIONS, AND FUTURE WORK

The joint creation process has limitations. For example, we might find that after reducing a panel edge to create space for a screw, there is not enough space left to mount a component placed on that panel. More robust methods for validating the joint creation is left for future work.

Implementing support for more advanced production techniques can make them available to a wider user group. A good example is the "kerf bending" technique (Figure 5c) which is hard to implement properly using drawing and CAD tools, but could be integrated in tools such as the one presented in this work.

We have not performed formal user evaluations of our system, but we have provided a proof-of-concept implementation of the design process of physical enclosures. For future work, we intend to perform user studies of our tool to make further improvements.

We focus on the static shape of enclosures, but realize that some enclosures also contain dynamic parts. In future work, we intent to support designing mechanical aspects as well.

In this paper, we used the .NET Gadgeteer platform as a case study. However, the idea of designing enclosures using electronic components is applicable to a broader array of prototyping frameworks. Other components such as Arduino, Sparkfun components or more generic micro-controller platforms could be used as well, provided their 3D models are available.

CONCLUSION

In this paper we presented a component centric interface for designing prototype enclosures. We use the components as handles for manipulating the enclosure shape. Our system targets laser cutting as production technique and handles production specific details, such as joint generation, automatically. We presented a proof-of-concept implementation of the system, including three prototypes that were designed and built with it.

ACKNOWLEDGEMENTS

This work was supported by the EU Marie Curie Network iCareNet under grant number 264738.

REFERENCES

1. Google sketchup. **`http://www.sketchup.com/`**.
2. Magic Box. **`http://magic-box.org/`**.
3. Tinkercad. **`https://tinkercad.com/`**.
4. Coffman, E. G., Garey, M. R., Johnson, D. S., and Tarjan, R. E. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing 9*, 4 (1980), 808–826.
5. FRONT. Sketch furniture, 2006. www.designfront.org.
6. Gross, M. Now more than ever: computational thinking and a science of design. *Japan Society for the Science of Design 16*, 2 (2007), 50–54.
7. Igarashi, T., Matsuoka, S., and Tanaka, H. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH* (1999), 409–416.
8. Landay, J. Design tools for the rest of us. *Communications of the ACM 52*, 12 (2009), 80.
9. Lau, M., Saul, G., Mitani, J., and Igarashi, T. Modeling-in-context: User design of complementary objects with a single photo. In *ACM Sketch-Based Interfaces and Modeling* (2010), 17–24.
10. Saul, G., Lau, M., Mitani, J., and Igarashi, T. SketchChair: An all-in-one chair design system for end-users. *TEI* (2011), 73–80.
11. Villar, N., Scott, J., Hodges, S., Hammil, K., and Miller, C. .NET Gadgeteer: A platform for custom devices. In *Pervasive* (2012), 216–233.