Editor: Mike Potel

Deco: A Design Editor for Rhinestone Decorations

Yuki Igarashi
University of Tsukuba

Young people enjoy applying rhinestones to personal goods, such as notebook PCs, mobile phones, and digital cameras. However, novices often find designing rhinestone decorations difficult because they must consider such factors as rhinestone width and spacing. So, many people employ off-the-shelf design sheets or have professionals decorate their items.

To solve this problem, I have developed an interactive editor for designing rhinestone decorations. Users draw free-form strokes; the system then automatically places virtual rhinestones to create an image in which no rhinestones overlap. Users can create a stencil of the image and use it to place the real rhinestones on objects. To see my system at work, watch the video "Deco" at www.geocities.jp/igarashi_lab/deco/index-e.html.

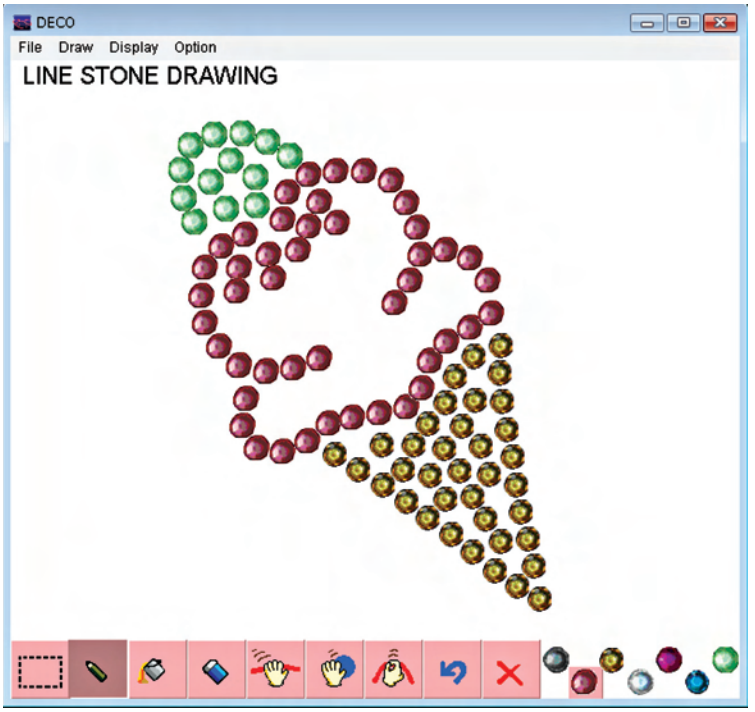

Figure 1. Deco's user interface. My system created this design on the basis of a user's free-form strokes.

Designing Rhinestone Patterns

Figure 1 shows Deco's interface. The system represents users' strokes as polylines so that users can easily move or delete rhinestones, change their colors, or change the image's scale. To delete a rhinestone or change its color, users right-click on it to open a pop-up menu (see Figure 2).

Users modify strokes using a pull tool (see Figure 3).[1] During a pull operation, the system interactively updates the rhinestones' positions and number.

Users can also upload existing images and place rhinestones over them to replicate the images (see Figure 4). This lets novices easily create designs.

To fill open spaces, the system offers two flood-fill modes (see Figure 5). In *grid-fill mode*, users apply sheets of rhinestones to quickly create attractive results. In *close-packed-fill mode*, users insert rhinestones one at a time to achieve evenly distributed results. Professional designers commonly use both design modes, but novices generally prefer grid-fill mode because it's easier to use.

Because total design times vary with each image, Deco estimates how long a design will take and displays the estimated time in a text box on the screen. The system also specifies the required number of rhinestones.

Producing the Finished Product

Once users have created a pattern, they can export it as a virtual stencil in vector file format; the system also supports the DXF format. Users can employ a cutting plotter such as CraftRobo (www.graphteccorp.com/craftrobo) to produce a physical stencil (see Figure 6a). They can then place the stencil over the object to be decorated and attach the rhinestones (see Figure 6b).

 Published by the IEEE Computer Society

Traditionally, professional users don't use stencils because they can design manually directly on an object (for example, a mobile phone, a hand mirror, or a T-shirt), but novice users generally find them useful. Although creating and implementing designs can be time-consuming, especially for beginners, I've found that most people enjoy it.

System Algorithms

I implemented my prototype system as a Java program that uses a vector representation.

Converting Input Strokes to Rhinestones

As Deco places rhinestones along a stroke, it samples candidate locations along the stroke at equal intervals L—a rhinestone's diameter, plus a small margin (see Figure 7). The system then determines whether the distance between each candidate location and any other rhinestones (except the previous one on the stroke) is greater than L. If this is so, the system places a new rhinestone.

Flood-Fill Modes

During flood-fill modes, when users click in a target region, Deco automatically places rhinestones there.

When a user clicks on a point $p_0 = (x_0, y_0)$, Deco places a rhinestone there. It then computes a *neighboring-vertices list* $N(p_0) = \{p_1, p_2, p_3, \ldots\}$ for p_0. In grid-fill mode, Deco computes four neighboring points—up, down, left, and right (see

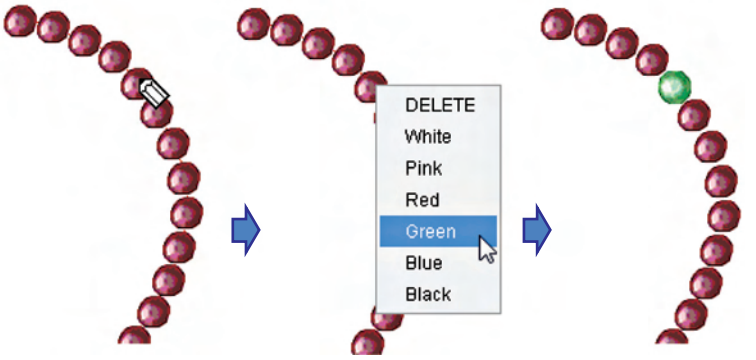

Figure 2. Right-clicking on a rhinestone opens a pop-up menu that lets users delete it or change its color.

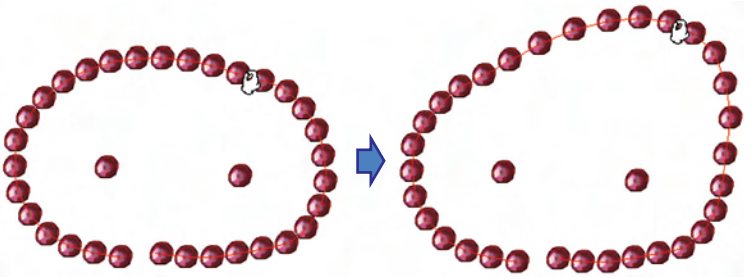

Figure 3. Users modify strokes using a pull tool. The system automatically updates the rhinestones' positions and number on each stroke.

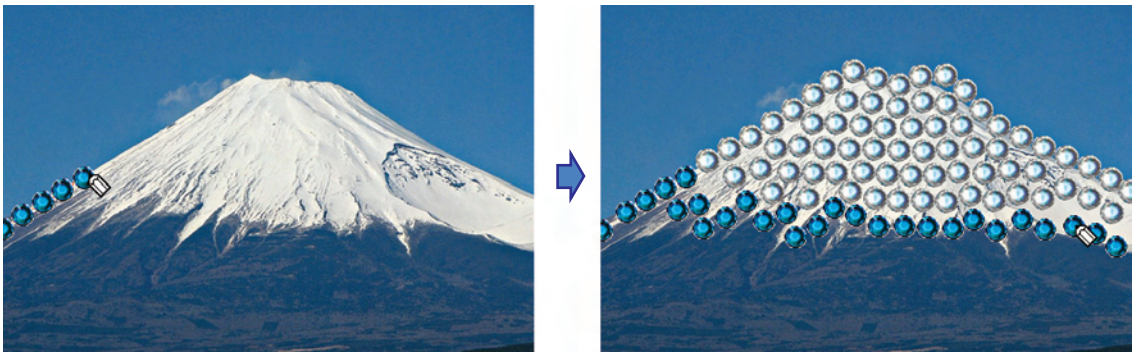

Figure 4. Users can upload existing images and place rhinestones over them. This lets novices easily create designs.

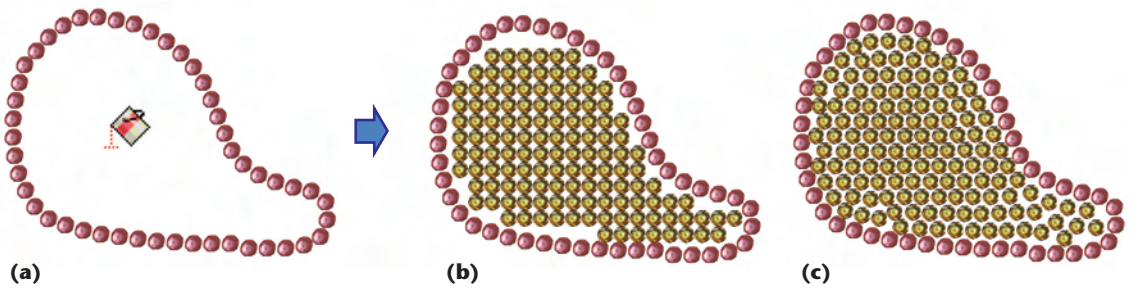

Figure 5. Flood-fill modes. (a) Users click in a field to use one of the modes. (b) In grid-fill mode, users apply sheets of rhinestones. (c) In close-packed-fill mode, users insert rhinestones one at a time. Novices generally prefer grid-fill mode.

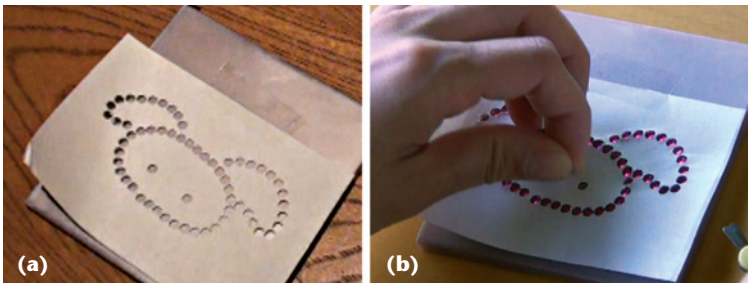

Figure 6. Creating the finished product. (a) Users make stencils using a cutting plotter. (b) The stencil's holes make placing rhinestones easy.

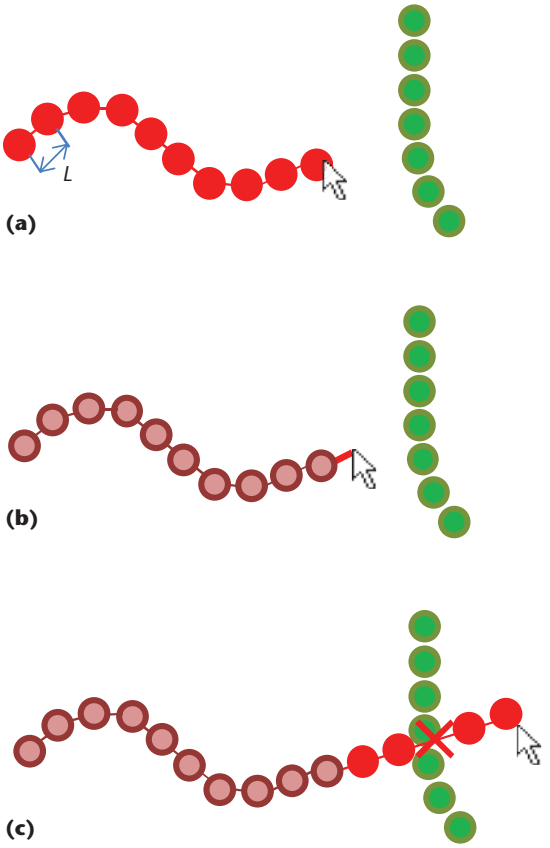

Figure 7. The algorithm for placing rhinestones on a user input stroke. (a) As users drag the mouse to create a design, the system automatically places rhinestones on top of the lines (b). When the distance between a mouse cursor and the nearest rhinestone is shorter than *L* (a rhinestone's diameter, plus a small margin), the system doesn't place a rhinestone in that location. (c) When the cursor passes over existing rhinestones, the system appropriately places new rhinestones.

Figure 8a) and adds them to the list. In close-packed-fill mode, Deco selects t evenly distributed points along the circle (see Figure 8b) and adds them to the list. I use $t = 6$ because it's the most densely packed configuration.

Then, Deco checks whether it can place a rhinestone at the six points it selected. If it can, it places the rhinestone and moves to a new position. If not, the system does nothing and waits for the user's next move.

In close-packed-fill mode, after placing rhinestones in a region, Deco adjusts their positions using the repulsion method. When the system computes the initial rhinestone fill results, it constructs an outside rhinestone list O and inside rhinestone list I. For each rhinestone $p_i \in I$, it evaluates

$$p_i^{k+1} = p_i^{k-1} + \sum_{p_j \in N(p_i)} \left\{ (10-k) \cdot \alpha \cdot e^{-\beta \left| p_i^k - p_j^k \right|_2} \cdot \frac{p_j^k - p_i^k}{\left| p_j^k - p_i^k \right|} \right\},$$

which iterates for $k = 0, 1, 2, \ldots, 9$. Deco's current parameter setting is $\alpha = 0.03$ and $\beta = 0.07$. In this equation, α and β are parameters to determine the expanding width, e is the exponential function $\exp(x)$, p_j is a position of neighbors of p_i, k is the number of iterates, and $N(p_i)$ is the list of neighboring positions. Deco repeats the repulsion process 10 times and moves each rhinestone. I call this a *quasi-close-packed* fill. The system can compute this in real time, which normally takes approximately 0.1 to 0.2 sec.

Total Estimated Production Time

The total production time includes cutting time plus decorating time. The cutting time (T_{cut}) increases in proportion to the rhinestones' total perimeter (L_{stone}). The total decorating time (T_{deco}) increases in proportion to the total number of rhinestones (N). So, the total time (T_{total}) is

$$T_{total} = T_{cut} + T_{deco} = W_{cut} L_{stone} + W_{deco} N,$$

where W_{cut} is the weight—that is, how long the cutting plotter cuts the paper per unit length—per unit L_{stone}, and W_{deco} is the decorating time for one rhinestone. I use $W_{cut} = 0.5$ sec./cm on the basis of CraftRobo's manufacturing information and $W_{deco} = 30$ sec. on the basis of user-study results. Users can easily reset W_{deco} to match their own speed.

Results

Deco runs in real time on a 1.1-GHz Pentium M PC; the design process generally takes 10 to 20 minutes, and production requires 30 to 60 minutes.

I ran a small workshop for novice users to try Deco. Ten children, ages 8 to 11, participated, accompanied by their parents (see Figure 9a). I began

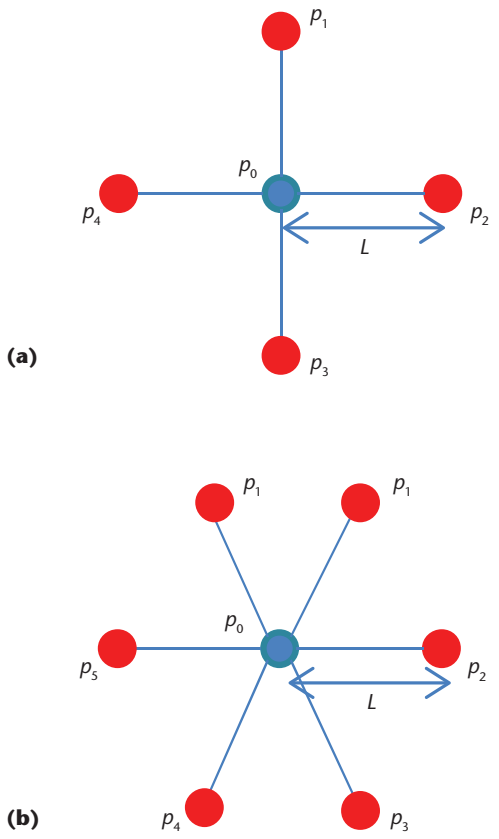

Figure 8. Calculating neighboring vertices. (a) Grid-fill mode. (b) Close-packed mode. The system constructs a rhinestone list in the field using the neighboring rhinestones. *P* represents rhinestone positions.

with a brief tutorial. The children took roughly an hour to design stencils (see Figure 9b), which they cut using a cutting plotter. The children then used the stencils to decorate accessory boxes, taking an additional hour (see Figure 9c).

Figure 10 shows some boxes that the children decorated. They quickly learned how to use Deco and enjoyed the workshop and design process. They were especially excited to see rhinestones generated on the fly as they drew the strokes.

The participants gave us valuable feedback for future improvements. For example, they agreed that using a stencil was helpful and that they'd like to create designs with greater detail—that is, design an animal's eyes, eyelashes, and mouth. One child noticed that the flood-fill method often failed to evenly distribute rhinestones because it inappropriately placed the boundary stones. So, he suggested that the system slightly adjust the boundary stones' positions to enable more even distribution.

The current implementation doesn't make allowances for rhinestones with different shapes. To enable this, I must extend the algorithm and develop a straightforward user interface for creating designs with rhinestones of different shapes and sizes. This would let users design in greater detail. Also, the system currently works only with a flat base; I plan to extend it to support bases with curved surfaces, such as a mug, a bottle, or an umbrella handle.

My research demonstrates the effectiveness of designing physical objects by considering the material's physical properties during construction.

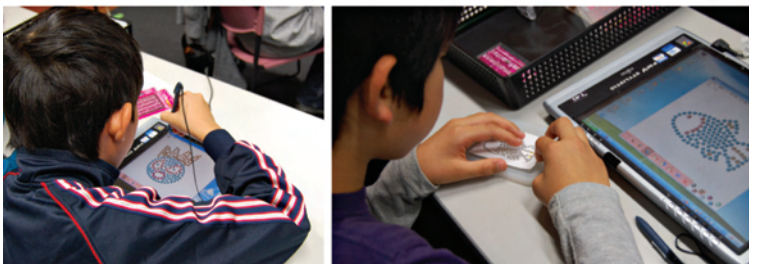

Figure 9. A workshop for novice users. (a) The participants. (b) A child designing a decoration. (c) Another child adding rhinestones to a box. The entire process took roughly two hours.

Figure 10. Some of the finished rhinestone decorations in the workshop. The children who participated in the workshop created these one-of-a-kind decorations.

On the basis of this philosophy, I've been developing design-support systems for other crafts, including stuffed or knitted animals.[2,3] I particularly aim to create systems that novices can easily use.

Acknowledgments

I thank the National Museum of Emerging Science and Innovation in Japan for providing the workshop location. A grant-in-aid from the Japan Society for the Promotion of Science Fellows partly supported this research.

References

1. T. Igarashi, T. Moscovich, and J.F. Hughes, "As-Rigid-as-Possible Shape Manipulation," *ACM Trans. Computer Graphics*, vol. 24, no. 3, 2005, pp. 1134–1141.
2. Y. Igarashi and T. Igarashi, "Designing Plush Toys with a Computer," *Comm. ACM*, vol. 52, no. 12, 2009, pp. 81–88.
3. Y. Igarashi and T. Igarashi, "Holly: A Drawing Editor for Designing Stencils," *IEEE Computer Graphics and Applications*, vol. 30, no. 4, 2010, pp. 8–14.

Yuki Igarashi *is a research fellow of the Japan Society for the Promotion of Science at the University of Tsukuba, Japan. Contact her at yukim@acm.org or www.geocities.jp/igarashi-lab.*

Contact department editor Mike Potel at potel@wildcrest.com.

ADVERTISER / PRODUCT INDEX • SEPTEMBER/OCTOBER 2011