Tearing instead of rectangular clipping/framing viewports in user interfaces

Janne V. Kujala and Tuomas J. Lukka
Hyperstructure Group
Agora Center, P.O. Box 35
FIN-40014 University of Jyväskylä
Finland
jvk@iki.fi, lukka@iki.fi

May 9, 2003

Abstract

We apply break lines from technical drawing as an alternative to the ubiquitous rectangular frames in user interfaces. When showing a piece of a larger canvas, the non-photorealistically torn edges naturally indicate where the content extends. We show how the shape of the jagged edge can be tied to its location on the paper; this creates a natural rippling motion when the edge moves in relation to the paper. The jagged edge is also visually less ambiguous than the straight horizontal and vertical lines and can be used to provide a sense of scale to the user.

We show how suitably rippling break lines can be implemented on the NV10 and NV25 architectures, and show screenshots of example user interfaces.

1 Introduction

In this article, we apply *break lines* or *break out section*s from technical drawing to *viewports* in computer user interfaces. Break lines or break out sections (see Fig. 1) are freehand lines drawn to indicate that an object extends beyond the part drawn in the diagram. These lines work by implying to the reader that it is *not* possible that the wavy line is actually the shape of the machine part, it has to be an artifact of the drawing.

Instead of framing a rectangular viewport to the canvas, we similarly "tear" a part of the canvas non-photorealistically, using break lines to indicate the torn edges.

In the following sections, we first describe related work, then the reasons and design issues and which features are desirable. Next, we describe a mathematical solution to the geometric problem and discuss a hardware-accelerated implementation. Finally, we discuss some example applications.

2 Related work

2.1 Viewports

As discussed in [23], viewports, i.e. regions of the screen (usually framed), showing part of a larger 2D plane (called *canvas* from here on), have been used in user interfaces since Sutherland's Sketchpad system[45].

Viewports are usually rectangular and parallel to the bounding rectangle of the canvas. (we shall not be concerned with occlusion by other graphical objects: we shall only concentrate on the basic characteristics of the viewport).

Even in systems that modify the conventional windowing model, the viewports appear to be mostly rectangular; for example in the 3D window manager Task Wall[42], Data mountain[40], Elastic Windows[19, 20],

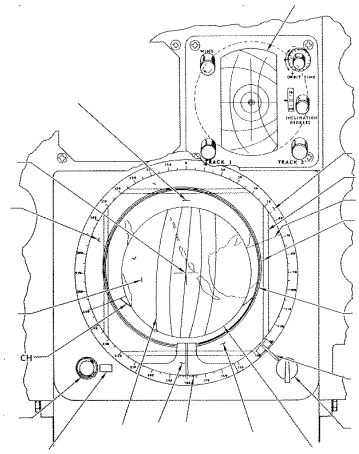


Figure 1: An example of break lines in a NASA drawing for the Mercury 5 (some text has been erased for clarity). Freehand lines have been long used in engineering drawings to indicate that an object extends beyond the shown part (that the part shown in the diagram is "torn" from its context).

3D WebBook[10], LifeStreams[13, 14], mUltimo3D[29], SPI hyperdocument presentation interface[17], PAD[37], its descendants PAD++[6, 4, 5] and Jazz[3], flip zooming[28], BookMap[18], the continuous zoom system in [2] and the Document Lens[41]. The Perspective Wall[31] simply folds the rectangular basic shape. As an extreme example of this, in [11], dealing with fisheye magnification, non-rectangular regions are magnified, but only rectangular regions are "lifted off" the original plane to become their own viewports,

This list is not intended as a criticism of the above work; we are only trying to demonstrate the dominance of rectangular, framed viewports. Indeed, the only references we found in the literature where non-rectangular viewports are actually used are [7] and [23]. In the toolglass system[7], a round magnifying glass is used. In Kramer's work on translucent patches[23], the non-rectangular windows are actually not viewports but rather complete regions, without a separate underlying canvas.

Non-rectangular window shapes *are* technically possible[35] but not used except for some special effects such as a round clock, and amusements such as shooting holes in windows or animated "agents"[1] such as the Office Assistants of Microsoft Office.

However, in all these cases, the irregular shape of the window matches the shape of an actual irregular object (magnifying glass, clock, shooting hole etc.). The irregular windows are not used as viewports, since they always show the whole irregular object, not only part of it.

2.2 Non-photorealistic rendering

Non-photorealistic rendering is a relatively new subfield of computer graphics. The principal ideas are presented by Saito and Takahashi[43]: creating images that emulate artistic drawings instead of photographs can clarify the images greatly.

Non-photorealistic rendering is able to focus user attention on the relevant details and also convey mood and semantic information (e.g., "this is not a finished design"[44]).

The recent work on non-photorealistic rendering has been concentrated more on rendering polygonal 3D models in ways which resemble paintings or drawings of real-world objects by human artists. However, we must point out that windowing systems are also usually non-photorealistic. Of course, this is not originally a UI design but a *technical* decision: lines were easier to draw than shaded surfaces.

The development of non-photorealistic algorithms and user interface components has been motivated by examples from illustration and technical drawing/design, for example silhouette and feature line rendering[43], cool to warm and metallic shading[16], and fillets[30].

3 Tearing

In this section, we introduce the use of non-photorealistic rough, torn shapes as break lines instead of the usual rectangular, clipped and framed viewports.

In the following subsections, we first discuss why tearing is desirable and how it differs from the usual viewports, then the details of the design and finally an algorithm with the required properties.

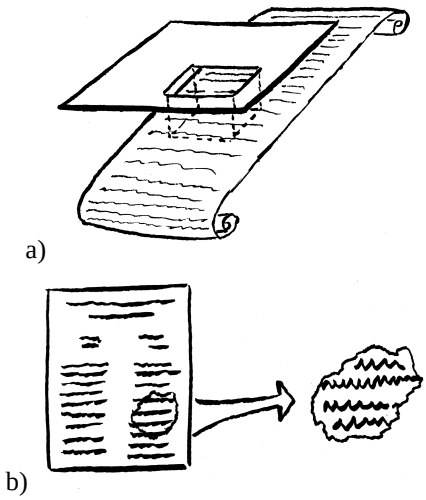


Figure 2: The metaphors: a) conventional viewports and b) torn viewports.

3.1 Rationale "Why?"

In a situation with several overlapping viewports (e.g. the type of focus+context view shown below), when the contents of the viewports have strong horizontal and vertical components, the irregular edges make it easier to understand where a viewport ends and where there is simply a line inside a viewport; see e.g. Fig.1.

One thing that we hope to achieve is that instead of having to perceive two objects, the "hole" of the viewport and the the canvas behind, the user would have to perceive only the torn piece, as in Fig. 2. Whether this can is achieved remains the subject of further study.

Additionally, the motion of the uneven edge can be used to give the user a cue about the motion of the viewport, and the style and size of the ripples can give a cue about the scale of the view (window frames have also been used for visualization previously[8]).

Now, from a purely physical perspective the tearout is not really a good metaphor[24]: a real torn piece of paper cannot change the place from where it is torn, with the edges rippling etc. However, the idea of break lines in technical drawings is well established and therefore the motion should be comprehensible.

3.2 Detailed design "What?"

In order to create and maintain the illusion: "we see a piece of the canvas", instead of "we see the canvas through an irregularly shaped hole", the motion must be carefully designed. The shape of the torn piece should not be translation-invariant but should change in an appropriate way with location and zooming.

To get the correct picture, imagine an animation where the first frame is a given torn piece of paper, the next frame is what would have happened if we had torn the paper slightly differently etc. Since the paper would have the same weak points, the shapes of the nearby tears would resemble each other. In particular, if the edge moves parallel to the (overall) direction of the edge, the shape should remain the same: see Fig. 3.

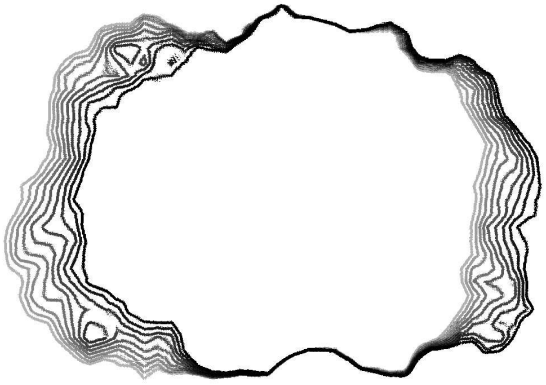


Figure 3: A diagrammatic representation of the motion of a torn viewport. The edges parallel to the direction of motion remain still while the moving edges change their shape.

As to the graphical appearance of the torn viewports, there are two main reasons to go for non-photorealistic rendering: firstly, to imply to the user that the viewport does not behave exactly like a real paper; to make the overall view clearer. Drawing only the silhouette edge[43] of the paper is therefore appropriate.

If the usual shape of the viewport extends outside the edge of the underlying canvas, the metaphor of tearing demands that we only draw the intersection of the two; see Fig xxx.

Finally, there are several design questions that are less important and which are best left as options in the algorithms, to be tailored to the particular application and taste. For example, whether the edge should only ripple without altering the topology or whether we should allow unconnected islands to appear (scattered edge); whether adding something to the roughly defined region to be viewed should always only move edges outwards or whether they may move back inwards at some point ("ebbing").

3.3 Algorithm "How?"

In this subsection, we formulate the design criteria of the preceding section mathematically and discuss simple algorithms for drawing shapes with the desired properties. The drawing of the silhouette lines is deferred to the later section on hardware-accelerated rendering.

We assume that we are given an undistorted shape A in paper coordinates, such as a rectangle or an ellipse, and we want to draw all pixels of B, the torn shape in paper coordinates. The transformation from A to B should, in accordance to the criteria in the previous section, be relatively local and depend continuously on the location with respect to the paper — but should not be translation-invariant.

The most obvious choice is to use an offset: if $a(\mathbf{p})$ is the indicator function for the shape A, i.e. 1 if $\mathbf{p}$ is inside A and 0 otherwise, then

$$b(\mathbf{p}) = a(\mathbf{p} + f(\mathbf{p})). \quad (1)$$

This is the way Perlin[36] create marble out of lines.

The limitation of this approach is that there is no ebbing because it is based on the points inside A, not its border. A different approach is to displace the border of A: if $\mathbf{a}(t)$ is the parametrized curve of the border of the original shape, then

$$\mathbf{b}(t) = \mathbf{a}(t) + f(\mathbf{a}(t)).$$

This technique is commonly called displacement mapping. There are variations to this such as displacing along the normal direction.

The important point w.r.t. both of of these common techniques is that the displacement depends on location in *canvas* coordinates, not in the coordinates relative to the undistorted shape. This is what produces the correct illusion.

4 Hardware-accelerated implementation

We shall concentrate on OpenGL and NVIDIA extensions (due to their availability in the Linux environment), but the feature sets of other manufacturers and proprietary APIs are quite similar.

OpenGL allows non-rectangular viewports through the stencil buffer, which can be used to create the stencil in a first pass by just drawing into the pixels and then in the second pass set to mask only those pixels to be allowed to be drawn that were touched in the first pass.

There are two basic alternatives for drawing the shape: either by using geometry to draw the jagged edge segment by segment, or by using a texture to draw a longer stretch at one time. We have chosen the latter approach as the more likely one to yield an acceptable performance. Also, it is easier to avoid aliasing artifacts in the texture approach. This approach, generating shape through texture, is similar to the one used by Perlin in [21] for synthesizing solid shapes.

4.1 Drawing the shape

When the tear-out reaches the edge of canvas, the smooth canvas border should be drawn instead of the parts of the tear-out shape extending outside the canvas. This can be implemented with stencil operations: First the full tear-out shape including the border is drawn to the stencil buffer. Then the canvas with its borders is drawn on screen using stencil test, and another bit is written to the stencil buffer for each canvas pixel ending up on screen. That stencil bit is then used to draw borders for the torn edges.

4.1.1 2D offset texture

The type of offsetting in Eq.(1) is implemented in modern texture shading hardware, such as the NV25 architecture. The image of the undistorted shape can be stored in a texture and accessed with texture coordinates offset by (read from) another texture. This is called an offset (dependent) texture access.

On hardware without texture shading capabilities, such as the NV10, we can obtain a suitable shape by constraining the offset in the direction of the normal. This can

be done by rendering into a texture a function which is 1 inside the undistorted shape and falls off linearly with distance. The hardware is capable of adding the value of such a texture and a noise texture and using that (through alpha test) as a condition for drawing the fragment.

We stress that this implementation is not necessarily any worse than the offset texture implementation: all its effects are within the options described in Section 3.2.

4.1.2 Explicit undistorted shape

Naturally, the undistorted shape can also be drawn by OpenGL primitives, by drawing polygons enveloping the widest variation in the torn shape and using the smooth shading interpolation in OpenGL to obtain the linearly falling value. This avoids the use of the second texture unit.

In this approach, we can also obtain ebbing with a kind of displacement mapping: instead of accessing the noise texture at the pixel location of the paper, the noise texture is accessed at the location of the undistorted edge by calculating the texture coordinates appropriately.

Corners can cause problems in this approach: for small curvature, the deviation from the normal direction is not noticeable but for angles approaching $90\,\mathrm{deg}$, the results are not satisfactory. There are at least two possible solutions: rounding the corners or using the stencil buffer to render the two edges separately and taking the intersection.

4.2 Drawing the edge

Most work to date on NPR silhouette edges concerns either polygon-based 3D models[9, 38, 32, 34, 39] or image-based approaches[43]. In our case, the edge is defined implicitly, as the solution of an equation, which makes it hard to obtain consistent border width. Mathematically, the border of an irregular tear-out is defined as the set of points whose distance from the tear-out is less than or equal to the desired line width. In the following, we consider different ways for approximating the border.

4.2.1 The oldest trick in the book

The oldest and conceptually simplest way to draw a black edge around a shape is to first draw the shape with black

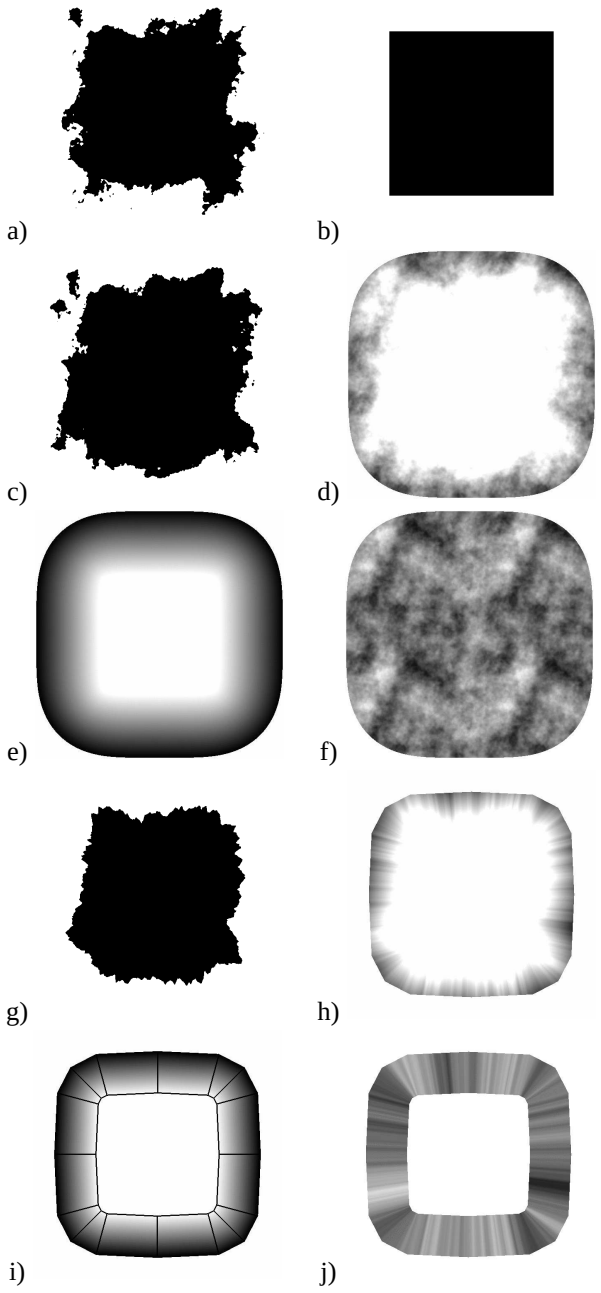


Figure 4: The algorithms a) 2D offset b) original texture that was offsetted c) 1D offset d) noise texture + undistorted shape texture e) undistorted shape texture f) noise texture g) explicit undistorted shape, ebbing case h) texture access results + color i) color j) texture access results

color several times shifted slightly to different directions before drawing the correctly colored, unshifted instance We have no idea when this approach has first been used or proposed.

Shifting to the four screen axis directions is simple and produces good results for narrow border widths (1 or 2 pixels). However, diagonal edges are drawn slightly too narrow, and the border around small features may end up sprinkled if the line width is more than one pixel. In such cases, a more complete set of shifted shapes is needed.

This method is general, but requires drawing the shape multiple times. With multiple texture units, it is possible in some circumastances to do the texture accesses corresponding to different shifts for each fragment at the same time.

Also, since we know that the center of the shape is solid, that part doesn't need to be drawn for the shifted shapes.

4.2.2 Pre-computed borders

Drawing thick (≈ 10 pixels) borders by shifting gets to be inefficient due to the large number of shifts required for good quality. In this section, we present an algorithm that is able to approximate the shape well in a single pass.

The algorithm works by precalculating the displacement or offset of the outer edge of the black line, assuming that the inner edge is defined as the edge of the area drawn by the algorithms of Section 4.1. The precalculated edge shapes are different for different angles, but can be approximated by storing the offsets at a discrete set of orientations in different components of a texture and interpolating by calculating dot products. This approximation is not completely free of artifacts, but is sometimes acceptable and is fast to draw.

Non-photorealistic line width scaling can be obtained by computing each mip-map[46] level of the outer edge textures separately with the desired line width for that scale. This method of computing mip-maps is similar to the art maps used in [22]. Because the textures store offset/displacement values, the mip-map levels interpolate seamlessly. However, zooming above the highest level of detail in the mip-map will fall back to the linear scaling.

4.2.3 Image-space algorithms

Image-space algorithms for drawing edges[43] work somewhat analogously to the previous section; the difference is that instead of rendering the shape several times, a filter is applied to the depth buffer where the shapes have been rendered to extract the discontinuities.

The advantage of image-space algorithms is that their performance does not depend on the complexity of a scene; however, it appears that only the NV30/R300 generation of graphics chips is flexible enough to support image-based operations well, due to the number of texture accesses and floating point operations needed. Also, it is more difficult (although possible) to adjust the width of the border based on depth in this approach.

4.2.4 A silly hack with offset texture mipmapping

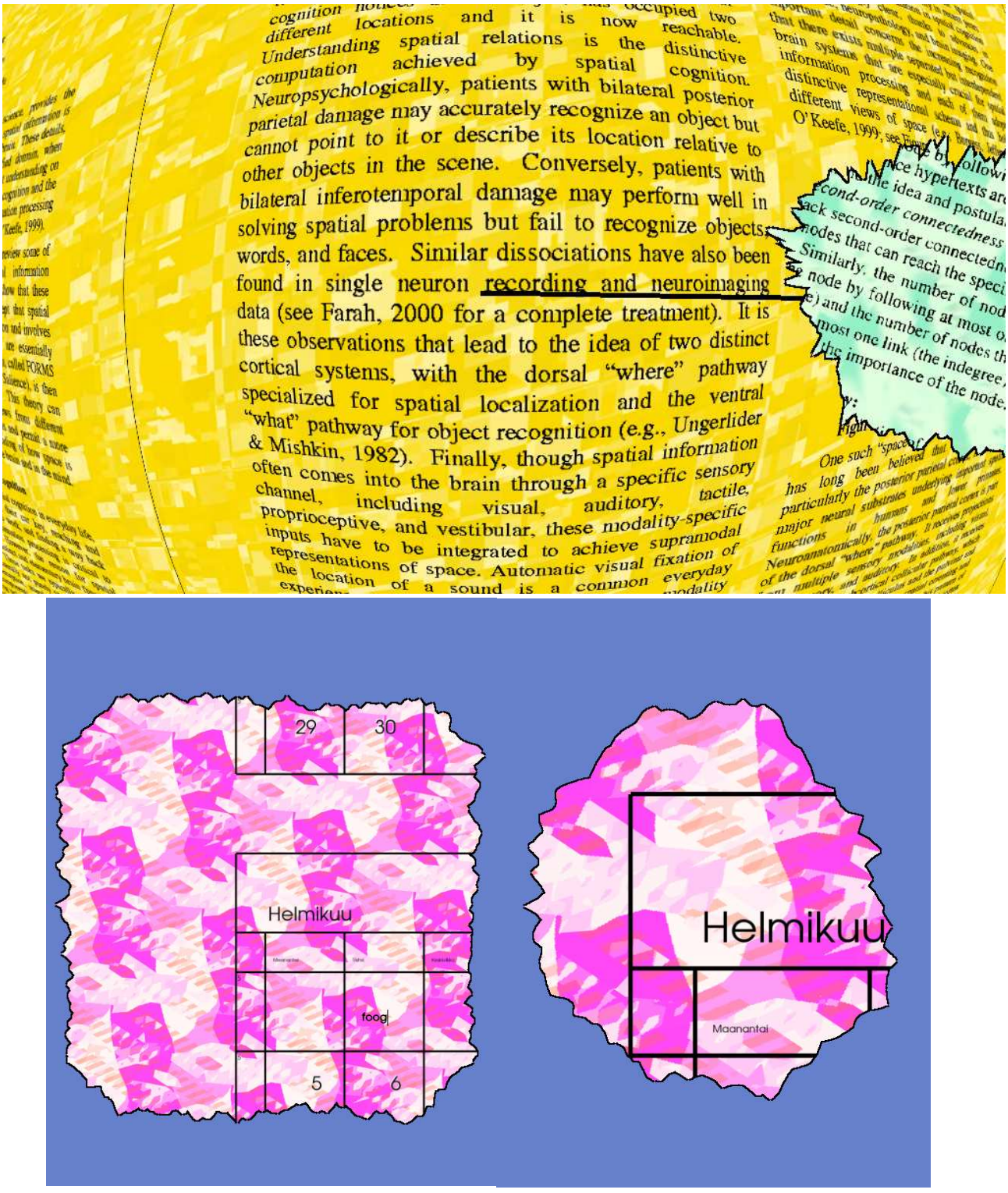
The border can be drawn by offsetting a texture with an image of a straight line. However, a sloped offset reduces the width of the distorted line. This narrowing can be compensated by using scale-invariant constant line width (in texels) for the mipmaps of the image of the line.

The trick is that the computed level of detail for each fragment (at least on the NV25) is lower for a sloped offset, and the lower detail texture with thicker line will exactly compensate the reduced line width. Note that this only holds when offsetting in the normal direction. Also, derivative discontinuities are sometimes visible as spikes in the border, if the border is more than a few pixels wide.

This trick only works if the line texture is 2D, but its other dimension can be 1.

5 Example applications

In this section, we discuss the screenshots of our user interface designs using tearouts (Fig. 5). The overall appearance is somewhat similar to the hyperbolic tree browsers[26, 25, 27]. The floating tearouts, which we call buoys, are somewhat similar to the child relation in the hypercept system of Milgram and Cowan[33] or the layout of links in the PAD++ web browser[4]. The difference is that the bidirectional links are elevated to first-class citizenship: the structure is not embedded to a 2D space before browsing. An example Also seen in Fig. 1 The connection structure somewhat similar to [12],



7

Figure 5: Screenshots of user interfaces using tearouts. Top: browsing PDF files of scientific articles, where the articles are bidirectionally connected. Each torn viewport represents a complete article to where the user can jump by clicking. The different background textures of the articles help the user know which article is shown in a tearout. the tearout. Bottom left: an interface where the user can change the zoom factor on the infinite plane. Bottom right: same view, zoomed. The ripples of the edge indicate the new scale, along with the background texture..

The interface is a focus+context[15] interface for browsing (editing is under construction) a structure of 2D planes between which there are bidirectional connections. The links connect particular regions of the planes, and the targets of the links are shown in tearouts. If a link target is clicked, the view animates, bringing the link target to full zoom and the current focus moves to a similar tearout.. For the link targets, rectangular frames would be less visually distinct; the tearouts shows clearly that they are not part of the focused document.

If the focused document is large, it is also shown with break lines, whose shape helps keeping track when panning and zooming.

We emphasize that this is only a prototype, about which we intend to publish a more detailed description later.

6 Conclusions

We have presented a new way of displaying viewports in computer user interface, based on break lines in technical drawings. The method can present a visually clearer appearance in situations with several viewports and also help perceiving viewport scale and motion.

http://linuxjournal.com/modules/NS-lj-issues/issue93/5574f3.png

6.1 Possible objections

One possible objection to the torn lines might be that they could take up more space than just the normal frames. However, the rounder shapes free some space, and in some cases, the uneven edge could even be seen as being used twice, i.e. providing *extra* space, because a wider context from both sides of the edge is shown, partially occluded.

Another objection is that the methods presented here are incompatible with some standard GUI elements and paradigms. For example, fitting scrollbars onto a tearout would be ridiculous. However, in a proper focus+context UI they should not be necessary: a click on the tearout would expand it and then the current location within the canvas could be shown in other ways, e.g. through a fisheye distortion. It is only natural that changing such a basic user interface element as the viewport will cause changes in the other elements that have adapted to the rectangular viewport for decades.

In its current form, the method is not suited for infinitely Zoomable User Interfaces (ZUIs)[37, 6].

6.2 Further work

The next generation of graphics chips brings interesting new possibilities with displacement mapping, floating point textures, longer fragment programs and more texture accesses per pass.

Carrying out usability tests adjust the parameters and to evaluate the potential of tearouts is a high priority.

7 Acknowledgments

The authors would like to thank XXX Canny!!, Benja Fallenstein and vegai and Antti-Juhani Kaijanaho for discussions.

References

[1] E. Andre, T. Rist, and J. Mueller. Employing AI methods to control the behavior of animated interface agents, 1998.

[2] Lyn Bartram, Albert Ho, John Dill, and Frank Henigman. The continuous zoom: a constrained fisheye technique for viewing and navigating large information spaces. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 207–215. ACM Press, 1995.

[3] B. Bederson, J. Meyer, and L. Good. Jazz: An extensible zoomable user interface graphics toolkit in java. In *UIST'00 Conference Proceedings*, pages 171–180, 2000.

[4] B. B. Bederson, J. D. Hollan, J. Stewart, D. Rogers, A. Druin, and D. Vick. A zooming web browser. In *SPIE Multimedia Computing and Networking 1996*, volume 2667, pages 260–271, 1996.

[5] Ben Bederson and Jon Meyer. Implementing a zooming User Interface: experience building Pad++.

Software Practice and Experience, 28(10):1101–1135, 1998.

[6] Benjamin B. Bederson, James D. Hollan, Ken Perlin, Jonathan Meyer, David Bacon, and George W. Furnas. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing*, 7(1):3–32, 1996.

[7] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: The see-through interface. *Computer Graphics*, 27(Annual Conference Series):73–80, 1993.

[8] Staffan Bj¨ork and Johan Redstr¨om. Window frames as areas for information visualization. In *Proceedings of the second Nordic conference on Human-computer interaction*, pages 247–250. ACM Press, 2002.

[9] J. Buchanan and M. Sousa. The edge buffer: A data structure for easy silhouette rendering, 2000.

[10] Stuart K. Card, George G. Robertson, and William York. The webbook and the web forager: An information workspace for the world-wide web. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'96*, 1996.

[11] M. S. T. Carpendale and Catherine Montagnese. A framework for unifying presentation space. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 61–70. ACM Press, 2001.

[12] Elizabeth F. Churchill, Jonathan Trevor, Sara Bly, Les Nelson, and Davor Cubranic. Anchored conversations: chatting in the context of a document. In *Proceedings of the CHI 2000 conference on Human factors in computing systems*, pages 454–461. ACM Press, 2000.

[13] E. Freeman and S. Fertig. Lifestreams: Organizing your electronic life, 1995.

[14] E. Freeman and D. Gelernter. Lifestreams: A storage model for personal data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(1):80–86, 1996.

[15] G. W. Furnas. Generalized fi sheye views. In *Proceedings of CHI '86*, pages 16–23. ACM Press, 1986.

[16] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. *Computer Graphics*, 32(Annual Conference Series):447–452, 1998.

[17] J. Hannemann, M. Thuring, and J. Haake. Hyperdocument presentation: Facing the interface, 1993.

[18] Mountaz Hascoet. A user interface combining navigation aids. In *Proceedings of the eleventh ACM on Hypertext and hypermedia*, pages 224–225. ACM Press, 2000.

[19] Eser Kandogan and Ben Shneiderman. Elastic Windows: Improved Spatial Layout and rapid multiple window operations. In *Proc. of the Workshop on Advanced Visual Interfaces*, pages 29–38. ACM, May 1996.

[20] Eser Kandogan and Ben Shneiderman. Elastic windows: Evaluation of multi-window operations. In *CHI*, pages 250–257, 1997.

[21] Eric M. Hoffert Ken Perlin. Hypertexture. *Computer Graphics*, 23(3):253–262, 1989.

[22] Allison W. Klein, Wilmot Li, Michael M. Kazhdan, Wagner T. Corrêa, Adam Finkelstein, and Thomas A. Funkhouser. Non-photorealistic virtual environments. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 527–534. ACM Press/Addison-Wesley Publishing Co., 2000.

[23] Axel Kramer. Translucent patches?dissolving windows. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 121–130. ACM Press, 1994.

[24] W. Kuhn. A formalization of metaphors and imageschemas in user interfaces, 1991.

[25] J. Lamping and R. Rao. Laying out and visualizing large trees using a hyperbolic space. In *Proceedings of ACM UIST*. ACM Press, 1994.

[26] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for viewing large hierarchies. In *Proceedings of CHI '95*. ACM Press, 1995.

[27] John Lamping and Ramana Rao. The hyperbolic browser: A focus + context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7(1):33–55, 1996.

[28] H. Lars. Focus + context visualization with flip zooming and the zoom browser, 1997.

[29] Jin Liu, Siegmund Pastoor, Katharina Seifert, and J?rn Hurtienne. Three-dimensional pc: toward novel forms of human-computer interaction. In *Three-Dimensional Video and Display: Devices and Systems SPIE CR76*, 2000.

[30] Tuomas J. Lukka, Janne V. Kujala, and Marketta Niemel¨a. Fillets: Cues for connections in focus+context views of graph-like diagrams. In *IV02 Conference Proceedings*, 2002.

[31] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: detail and context smoothly integrated mackinlay color plates. In *Human factors in computing systems conference proceedings on Reaching through technology*, pages 173–176. ACM Press, 1991.

[32] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-time nonphotorealistic rendering. *Computer Graphics*, 31(Annual Conference Series):415–420, 1997.

[33] Dan Milgram and William B. Cowan. Hypercept: behavioural linkage in hypertext environments. In *Proc. NPIVM'99*, pages 49–55. ACM Press, 1999.

[34] J. Northrup and L. Markosian. Artistic silhouettes: A hybrid approach, 2000.

[35] Keith Packard. X nonrectangular window shape extension protocol. X Consortium Standard, MIT X Consortium, 1989.

[36] K. Perlin. An image synthesizer. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):287–296, 1985.

[37] Ken Perlin and David Fox. Pad: an alternative approach to the computer interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 57–64. ACM Press, 1993.

[38] R. Raskar. Hardware support for non-photorealistic rendering, 2001.

[39] Ramesh Raskar and Michael Cohen. Image precision silhouette edges. In *Proceedings of the 1999 symposium on Interactive 3D graphics,Atlanta, Georgia, United States*, pages 135–140, 1999.

[40] George G. Robertson, Mary Czerwinski, Kevin Larson, Daniel C. Robbins, David Thiel, and Maarten van Dantzich. Data mountain: Using spatial memory for document management. In *ACM Symposium on User Interface Software and Technology*, pages 153–162, 1998.

[41] George G. Robertson and Jock D. Mackinlay. The document lens. In *Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 101–108. ACM Press, 1993.

[42] George G. Robertson, Maarten van Dantzich, Daniel C. Robbins, Mary Czerwinski, Ken Hinckley, Kirsten Risden, David Thiel, and Vadim Gorokhovsky. The task gallery: a 3d window manager. In *CHI*, pages 494–501, 2000.

[43] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 197–206. ACM Press, 1990.

[44] Jutta Schumann, Thomas Strothotte, Stefan Laser, and Andreas Raab. Assessing the effect of non-photorealistic rendered images in cad. In *Conference proceedings on Human factors in computing systems*, pages 35–41. ACM Press, 1996.

[45] Ivan Edward Sutherland. *Sketchpad, a man-machine graphical communication system*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering, 1963.

[46] Lance Williams. Pyramidal parametrics. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 1–11, 1983.