VisualIDs:
Automatic Distinctive Icons for Desktop Interfaces

Supplementary Material

J.P. Lewis[*]
CGIT Lab
U. Southern California

Ruth Rosenholtz
Perceptual Science Group
Massachusetts Institute of Technology

Nickson Fong
ESC

Ulrich Neumann
CGIT Lab
U. Southern California

[*]zilla@computer.org

Shape Grammar

The complete parameters for the shape grammar are described here. In the parameter descriptions:

- `level` is the recursion level, starting at 1 and increasing with smaller scale detail.
- `Rnd11()` returns a uniform pseudo-random value in -1,1.
- `rndinLowbias(a,b,p)` is a version of the low-bias generator where the power can be specified: `a + (b-a)*pow(Rnd(),p)`.
- `rational(lo,hi,den)` returns a rational number (cast to real) where the numerator is in `lo,hi` and the denominator is `den`.
- `rndprobLevel(p)`: The probability is reduced at each level using `p = p - p*min(level,5)/6`
- `rndinLevel(a,b)`: the range is reduced by recursion level, i.e., `a + Rnd()*(b-a)/level`.

Around-a-spiral

`len`	`rndin(25,60)`	number of segments in the spiral (if level=1)
`len`	`rndin(10,30)`	number of segments in the spiral (if level=2)
`len`	`rndin(5,10)`	number of segments in the spiral (if level=3)
`len`	`5`	number of segments in the spiral (if level $>$ 3)
`aligned`	`rndprob(0.5)`	children are aligned on radial spokes
`ppr`	`rndinLowbias(3,35)`	points per revolution (if aligned)
`ppr`	`rational(3,35, rndin(3,7))`	points per revolution (if not aligned)
`hasChild`	`rndin(0.5,1)`	place a child generator at points along the spiral
`centerScale`	`rndin(0.05,0.6)`	scale reduction for the child at center of spiral
`outerScale`	`rndin(0.25,1)`	scale reduction for the child at outside of spiral

The spiral algorithm is

```
for( i=0; i < len; i++ ) {
  theta = 2*pi*float(i) / ppr;
  r = 0.15 * sqrt(theta);
  x = r * cos(theta);
  y = r * sin(theta);
}
```

The `ppr` is adjusted so that there are at least two revolutions in the spiral.

Around-a-Shape and Relaxed Inside

Both of these are implemented with the same code, with a boolean variable `hasOC` controlling whether a child is generated around the outline.

The outline is created with either a Fourier series or FFT filtering of noise, according to the variable `fourier`. Fourier series generation produces a more regular or coherent outline. Pseudocode for the Fourier series is

```
ncoef = 2*rndin(2,5) + 1;
for( i = 0; i < ncoef; i++ )
  amp[i] = Rnd11() / float(i+1); // 1/f envelope
```

`n`	`rndinLevel(3,6)`	number of objects (power)
`fourier`	`rndprob(0.5)`	use fourier series (else FFT)
`exponent`	`rndin(1.6,2.2)`	spectral exponent for FFT filtering
`hasIC`	`rndprobLevel(0.6)`	a child is placed inside the outline
`hasOC`	`rndprobLevel(0.6)`	a child is placed around the outline
`OCscale`	`rndin(0.1,0.6)`	scale reduction for the outline child

The number of interior children is 2^n, and each child is scaled down by rndin(0.1,2)/2^n.

Along a Path

The outline is created with either a Fourier series or FFT filtering of noise, according to the variable `fourier`. The code is similar to that for the `relaxed inside` generator, except that a subrange of the synthesized outlines between parameter 0,0.7 is used to produce an open, aperiodic curve, whereas `relaxed inside` uses the full 0,2*pi range of the outline.

The number of children placed along the curve is 2^n.

`hasChild`	`rndproblevel(0.8)`	place child along the path
`Cscale`	`rndin(0.1,0.6)`	child scale reduction
`n`	`rndinLevel(3,6)`	(power) number of children placed along the path
`fourier`	`rndprob(0.5)`	use Fourier series synthesis
`exponent`	`rndin(1.6,2.2)`	spectral exponent for FFT filtering

Scribble

The `scribble` generator traces a path attracted by random or patterned attractors and damped by “friction”. In the random case attractors are randomly distributed throughout the unit square centered on the origin. Patterned attractors `X[i],Y[i]` are distributed in a “zig-zag” pattern according to the code:

```
float dx = 0.f;
float dy = 1.f;
float cx = 0.f;
float cy = -0.5f;
float opposingRatio = rndin(0.05f, 0.2f);
for( int i=0; i < ncvs; i++ ) {
  X[i] = cx;
  Y[i] = cy;
  if (i%2 == 0) {
    cx = cx + dx;
    cy = cy + dy;
  }
  else {
    cx = cx - dx;
    cy = cy - dy;
  }
  // move perpendicular
  cx = cx - opposingRatio * -dy;
  cy = cy - opposingRatio * dx;
} //for
```

The attractors are applied in sequence, with a new attractor taking effect when the line has approached a preset distance (0.03) from the current attractor. The attractor force is in the direction from the current point to the attractor. The scribble algorithm is best described in code:

```
float jitter = 0.1f;
float dx = X[1] - X[0];     // X,Y are the attractor locations
float dy = Y[1] - Y[0];
dx = dx + jitter * rndf11();
dy = dy + jitter * rndf11();
float len = (float)sqrt(dx*dx + dy*dy);
float distthresh = 0.03f;
float force = 0.01f;
float cx = X[0];
float cy = Y[0];
float vx = 0.f;
float vy = 0.f;
int ncvs = X.length;

moveto(cx, cy);             // use moveto/lineto drawing

for( int icv=0; icv < ncvs; icv++ ) {
  int icv1 = (icv + 1) % ncvs;
  float spandx = X[icv1] - X[icv];
  float spandy = Y[icv1] - Y[icv];
  float dot = 1.f;
  boolean flag = true;
```

```
  while(flag) {
    // integration
    force = (len > 0.2f) ? (force*1.1f) : (force*0.7f);
    if (force > 0.08f) force = 0.08f;
    if (force < 0.01f) force = 0.01f;
    vx = vx + force * dx;
    vy = vy + force * dy;
    vx = vx * _friction;
    vy = vy * _friction;
    cx = cx + vx;
    cy = cy + vy;
    lineto(cx, cy);

    // new direction
    dx = X[icv1] - cx;
    dy = Y[icv1] - cy;
    len = (float)sqrt(dx*dx + dy*dy);
    dx = dx + (float)abs(dx)*jitter*rndf11();
    dy = dy + (float)abs(dy)*jitter*rndf11();

    // end segment?
    dot = spandx*dx + spandy*dy;
    if (len < distthresh)
      // near attractor, break
      flag = false;
  } //while(flag)
} //icv
```

`zigzag`	`rndprob(0.5)`	zig-zag patterned impulses (else random)
`n`	`rndin(2,30)`	if !zigzag and level=1
`n`	`rndin(10,20)`	if !zigzag and level>1
`n`	`rndin(40,10)`	if zigzag
`friction`	`rndin(0.85,0.98)`	if !zigzag
`friction`	`rndin(0.8,0.9)`	if zigzag

Symmetry

The symmetry generator draws nothing itself but applies its child in an n-lateral radial symmetry. If n=2 the child is mirrored.

`n`	`rndinLevelLowbias(2,8)`	n-lateral symmetry
`offset`	`rndprob(0.3)`	translate out from center
`offsetTrans`	`rndin(0.1,0.5)`	amount to translate out from center
`Cscale`	`rndin(0.3,0.6)*(1-offsetTrans)`	child scale reduction

Line generator

Although `line` serves as a terminal in the grammar we also use it as a non-terminal with these parameters:

`len`	`rndin(0.5,1)`	length of the line
`n`	`rndinLowbias(3,10,1.5)`	number of C3 children along the line
`hasEC1`	`rndprob(0.75)`	has child C1 at end 1
`hasEC2`	`rndprob(0.75)`	has child C2 at end 2
`hasC3`	`rndprob(0.5)`	has child 3 (along the line)
`C1scale`	`rndin(0.15,0.6)`	scale reduction for child 1
`C2scale`	`rndin(0.15,0.6)`	scale reduction for child 2
`C3scale`	`rndinLowbias(0.05,0.7,1.5)`	scale reduction for child 3
`C3doublesided`	`rndprob(0.8)`	child 3 is drawn mirrored on both sides
`C3angle`	`rndin(-pi/4,pi/4)`	angle of child3 relative to line normal
`C3taper`	`rndprob(0.3)`	reduce or increase scale of child3 by 1.5 along the line
`C3increase`	`rndprob(0.3)`	increase (rather than decrease) child3 scale along line

Figure generator

The `figure` generator produces an animal-like arrangement of generators. Most generators are forced to be `line`, while the “head” is allowed to be a `line` or `radial`.

`headScale`	`rndin(0.1,0.2)`	scale reduction for the head child
`legScale`	`rndin(0.3,0.6)`	scale reduction for the leg children
`tailScale`	`rndin(0.1,0.4)`	scale reduction for the “tail” child

FAQ

This paper has been read by a number of people and we have received lengthy comments about it, many of which refined our presentation. This section covers some of the objections that were raised.

Icons do not generally qualify as scenery. Our definition is not too different than the one in WordNet (which is evidently derived from actual usage): "the appearance of a place." While some places (such as a beach) are difficult to divide into meaningful "objects", others (such as a city, or most other man-made environments) are clearly composed of objects in the everyday sense. The scenery in a city is *the appearance of its non-moving objects* (buildings, etc.). If all the buildings in your town had a uniform size, shape, and color, you would have a lot more difficulty finding your way!

This "object-based" definition of scenery is in fact *necessary* for files, since files are discrete objects. The form of scenery you are probably thinking of (more like the beach example) would take the form of distinguishable appearance for the (folder) backdrop. This is also desirable and we advocate it in the paper as well, but it is not as essential. For example, if every folder had a distinctive picture as its backdrop when opened, but the file- (object-) based scenery was absent, navigating to other folders and finding files would still be difficult, since all sub-folders and files would have a generic appearance.

The experiment's relationship to the larger issues of 3D navigation are not immediately apparent. We do not claim that this paper directly addresses navigation. But, navigation in featureless spaces is difficult and adding distinctive appearance is known to help navigation ([Searleman and Herrmann 1994; Ruddle et al. 1997] and to a lesser extent [Ingram and Benford 1995; Darken and Sibert 1996; Robertson et al. 1998]). Scenery as we define it *complements* spatial navigation schemes and is *required* to engage our visual brain in navigation.

People use natural languages to express information much more frequently than we use pictures. If people could not even remember the labels given in natural language, it's hard to believe that they can remember graphical labels better than the textual labels. Both psychological and HCI evidence (Section 1,2) and our own studies 1,2 show the opposite.

There are not enough pixels on current screens to show all these icons. The icons used in the studies are 64^2 (at least one current OS has icons of this size) and are quite distinguishable. Further, the idea of zoomable interfaces (e.g. [Perlin and Fox 1993]) is well established and could be used if necessary to distinguish very similar icons such as those resulting from mutation.

You argue that the icon does not need to be related to the content. People normally remember the content but not the exact labels of the documents. I just don't know how the content-irrelevant icons can help people recall. It is true that people remember the content of documents. Documents also have labels, and it's true that people often do not remember the exact labels. The comparison here is between textual labels and textual+visual labels. All the evidence (prior and ours) supports the idea that humans have strong abilities to find and recall by appearance, and when given the choice, we try the visual approach first [Byrne 1993]. For example, I locate one of my books (having some desired content) by appearance, not by reading the title of each book in my collection.

How does the system treat the case in which a set of files is named similarly (paper1.tex, paper2.tex, ...) but one differently (final.tex) which, however, is similar in content but unfortunately not in the naming. The `paper1.tex, paper2.tex, ..., final.tex` issue was addressed in the **Similar Identity versus Similar Content** section on page 43 This is a good example of what was described there: although the content of all these files are similar (`paper1, paper2,` ... being successive versions of the paper), their *meaning* for the user is different. And this difference in meaning can be seen from the file name but not the file content! The user names the final version to `final.tex` to make it stand out from the others; it is the valuable final version, whereas the others are backup versions. In the mutation proposal its file icon will also stand out from the others, as desired.

What if the users change filenames often or even reorganize their file system. Then an icon that has been the visual hook for a file will be newly created and differ from the previous one. Would not the user be even more lost then? No. Users can certainly loose files by moving and renaming them, regardless of the current proposal. See the **Coexistence, not Replacement** section: the original file names are still there; the scenery is just one additional cue. As all the previous information is still there, no one should be "more lost". The act of renaming a file has a non-trivial "cost" – the user must remember the new name. The evidence indicates that this remembering task is faster and more robust with appearance data than it is with only textual labels.

The other question suggested here is whether it is best to change the icon for a file when the name changes. By the argument above, neither possibility should really hurt performance, but perhaps one choice will improve performance more than the other. Although this question probably needs further study, we suspect that changing the icon along with the name is the most consistent choice. In the `paper1.tex, paper2.tex, ..., final.tex` example, we suppose that the user names or renames the final version `final.tex` to make it stand out from the others. If they are renaming the final version, and the icon does not also change, the file's appearance will be misleading – it will look like one of the earlier versions.

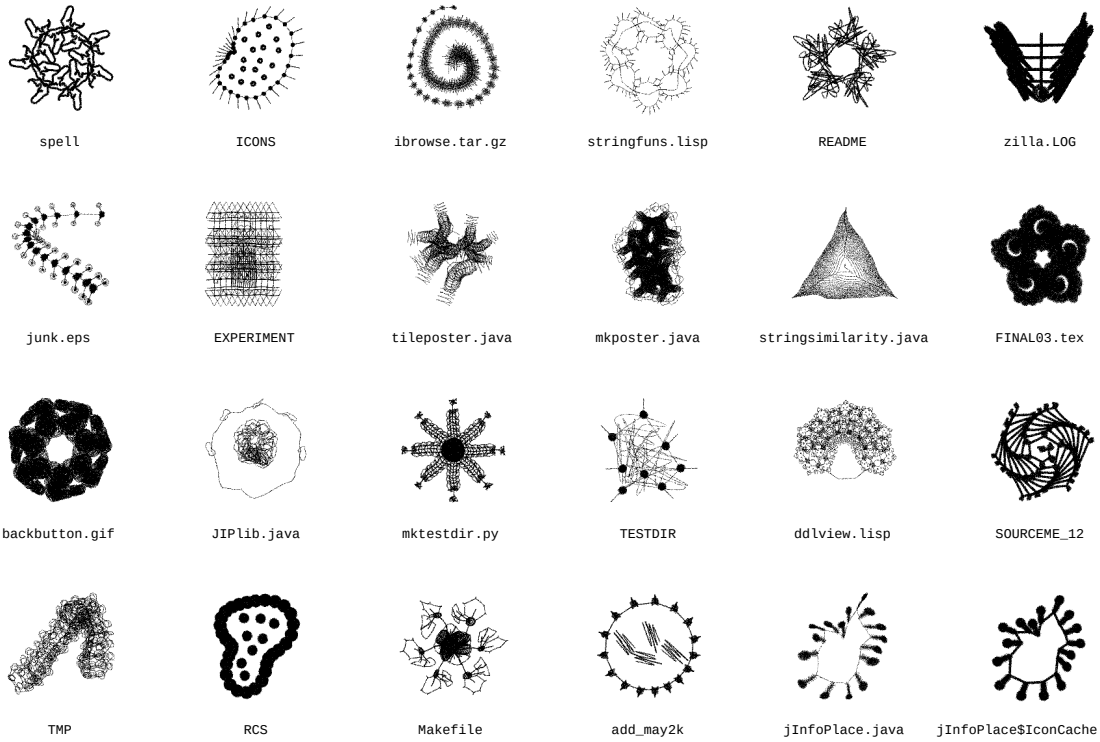
If I rename a number of files and their icons all change, how can I ensure that the new icons bear the same relationship to each other that the old ones did? Probably this is not possible, but why is it necessary? There is nothing indicate that we would be particularly bothered by this scenario, on the contrary, it is a visual learning problem of the sort that we are quite skilled at. In the real world, my favorite bookstore and the adjacent coffee shop may well be replaced by a Starbucks and a 7-11, which do not bear the same relationship as the original two stores... but I'll have a lot more difficulty remembering a mathematical fact, or even someone's name, than the appearance of the new stores.

Photographs versus abstract icons. Small photographs have many of the same benefits as abstract scenery, and are likely to be more distinguishable than abstract icons. On the other hand there are reasons to prefer abstract icons: 1) the use of similar icons for similar filenames might be difficult or impossible with photographs; 2) algorithmic construction offers additional possibilities for branding and query; 3) abstract icons are relatively (though not entirely) free of the semantic and culturally specific issues of interpretation that can interfere with

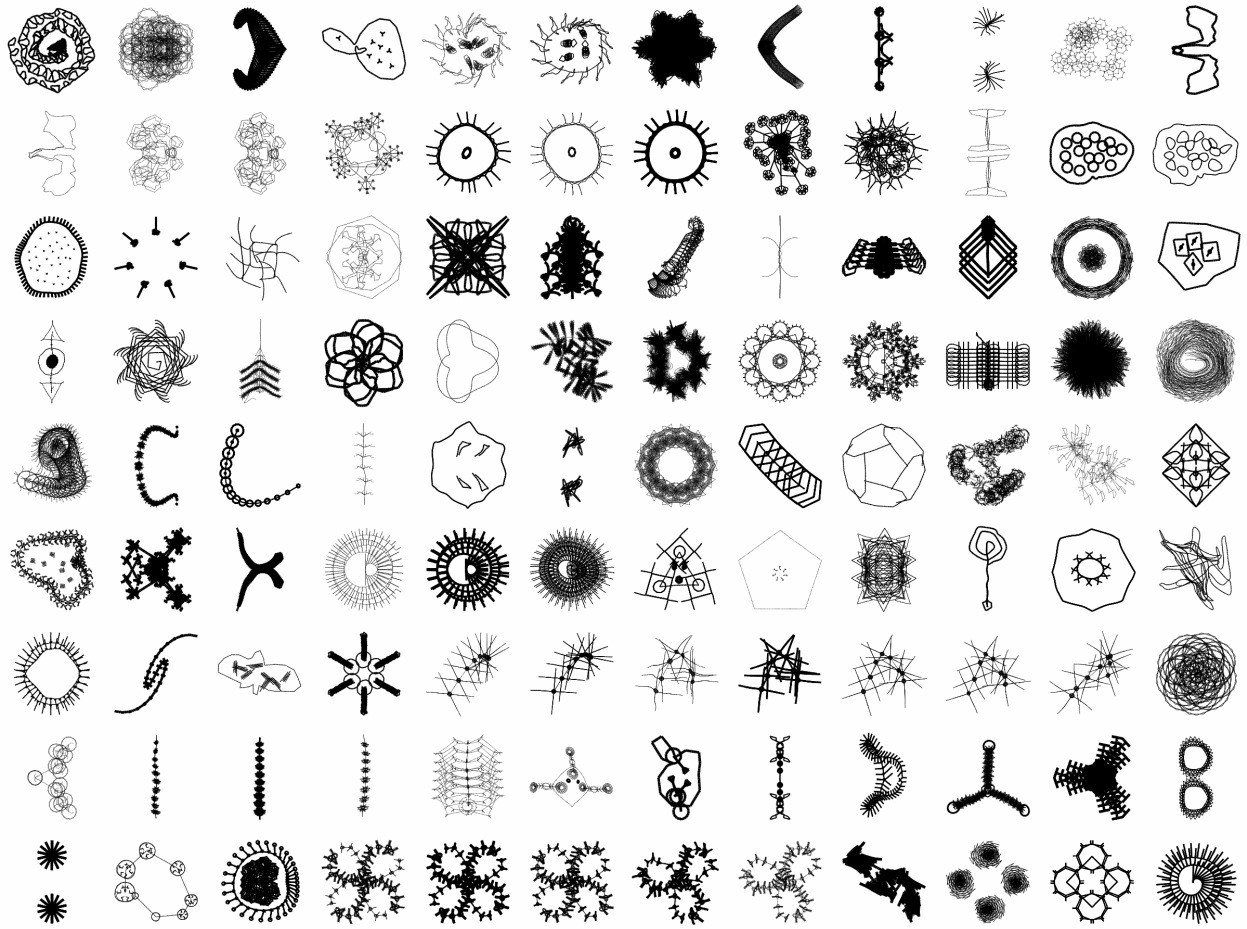
icon success (A. MacEachren, *How Maps Work: Representation, Visualization, and Design*, Guilford Press, NY, 1995).

It slightly worries me that even with the current scheme, producing a workable scenery construction requires both artistry on the part of the programmer and tuning. Some existing GUIs similarly involved skilled programmers and designers.

You did not consider [issue X]. Yes, we did not consider all issues that arise from this "scenery" idea. A single paper on most any topic cannot cover every related issue, and admittedly this paper has more unresolved (and interesting) issues than most.



More doodle-style file icons. The maximum recursion level is increased to generate more detailed icons. Clustered filenames appear together, but overall order is arbitrary due to traversing a hash table.



More doodles.

Browser prototype screenshot

Additional details on the user studies:

To eliminate the time required for generating icons dynamically
a set of pre-generated icons was used in the experiments.
Filenames were taken from the project directory for
this paper and icons were assigned to them arbitrarily (files
ordered by the java File.list() call were matched with icons
in order of generation).

There was a precursor to study 1.
In this version subjects (professional computer artists)
were instructed to find particular files in a mock interface
displaying 200 files in a scrollable window.
There were two conditions, with- and without-doodle icons.
Eight files were requested twice each,
thus testing for spatial and icon memory.
While the total search time for the ‘scenic’ condition was
slightly shorter than under the generic icon condition
(and was mildly significant) we decided to revise this study based
on user feedback. The comments were that:
1) a single repetition was not enough
2) much of the elapsed time was spent on the mechanics of scrolling
3) in both conditions subjects reported that they
made use of the spatial layout in finding files-
for example they might recall that a particular file
was somewhere to the upper left in the scrollable display.

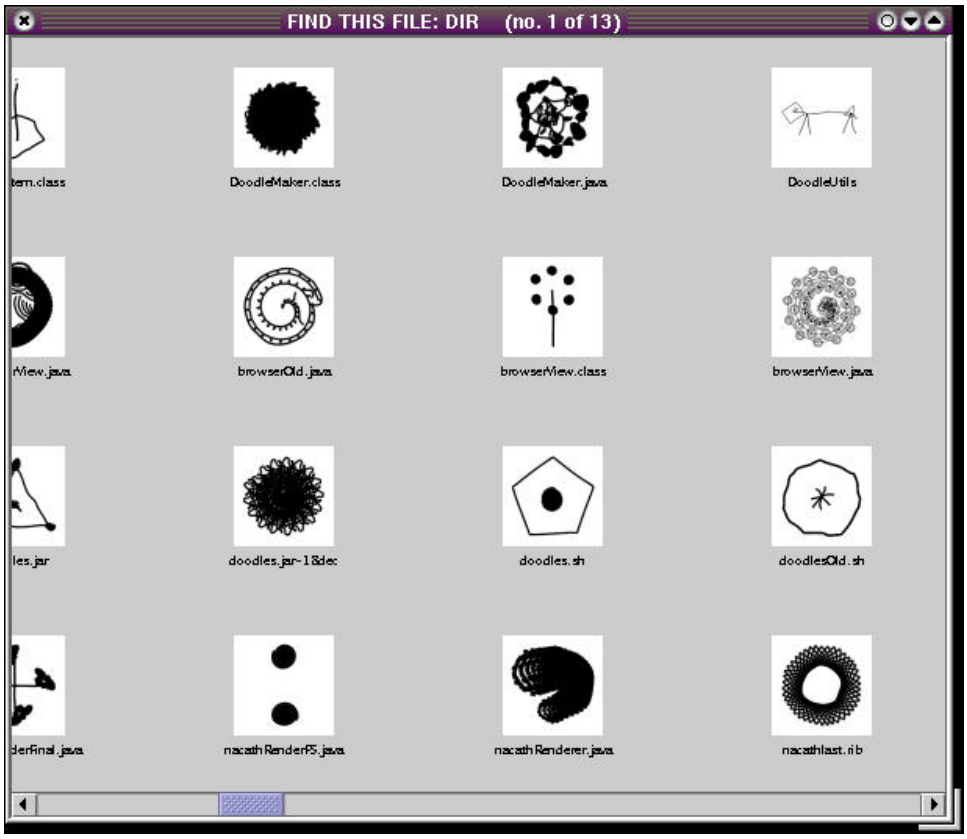
These comments suggest that the study design
did not clearly show the potential effect of distinguishable icons.
Simply running this study for a longer period (more than one repetition)
would be a suitable fix, but we wanted to keep the total time
to a ‘‘polite’’ 5-10 minutes.

In study 1 the number of files was reduced to 24 --
this is probably a representative number for a single folder.
This study was implemented as a java browser client
that uploads results to a multithreaded server.
Source for this and the second study is available at
http://www.idiom.com/~zilla/Work/VisualIDs/Expcode
(note that studies 1,2 are referred to as Experiment 4
and Experiments 5,6 in this code).

Participants were recruited by email.
56 people started the study but 26 completed it despite
the relatively short length. The high number of incomplete
results was probably due to a java/browser incompatibility
that caused the applet to crash when accessing the icon list
(but after opening the server connection). This bug was
fixed before the second study.
One result was discarded as the elapsed time was several hours
(the user presumably was interrupted and returned to complete
the study later).
Screenshots of Studies 1 and 1-precursor are shown below.

The second study used the same
browser/server approach developed for study 1.
41 people did part or all of day 1,
21 completed both days, of these 9 were in the with-icon condition.
The participants were split among graduate students and
professional computer artists. The large number of incomplete results
in this study was probably due to it being a more difficult
multi-day study.

In followup comments some users in the generic condition said
‘‘how were we supposed to know [what to answer]’’, meaning, the idea

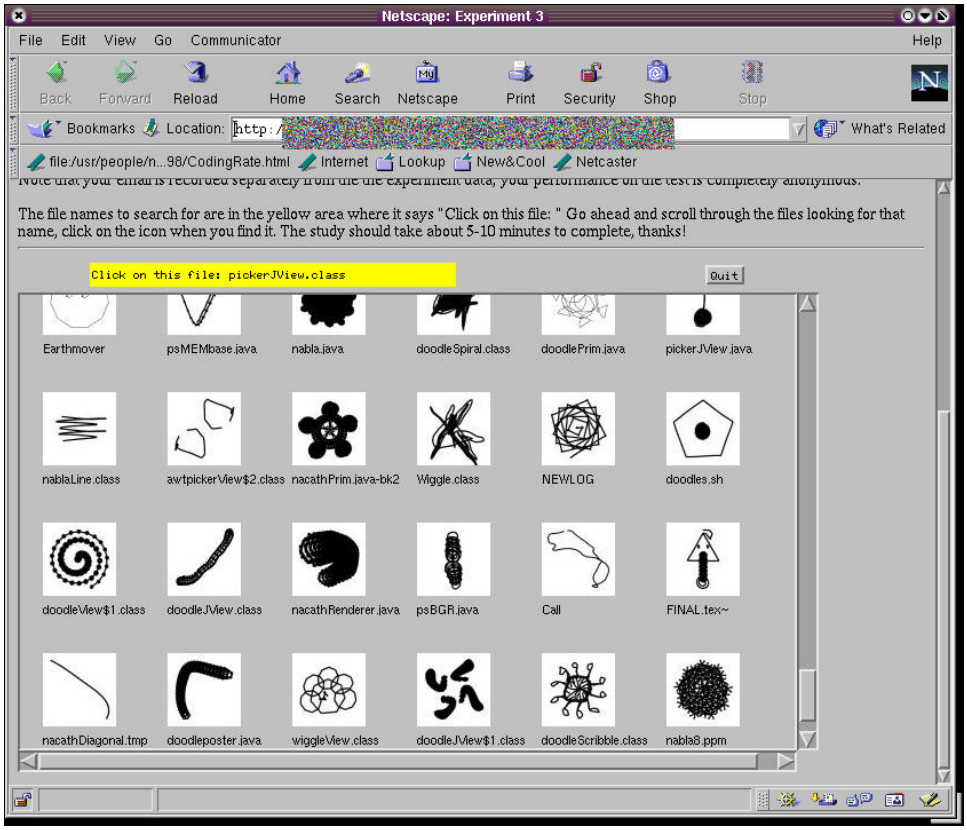


Screenshot of study 1 precursor.

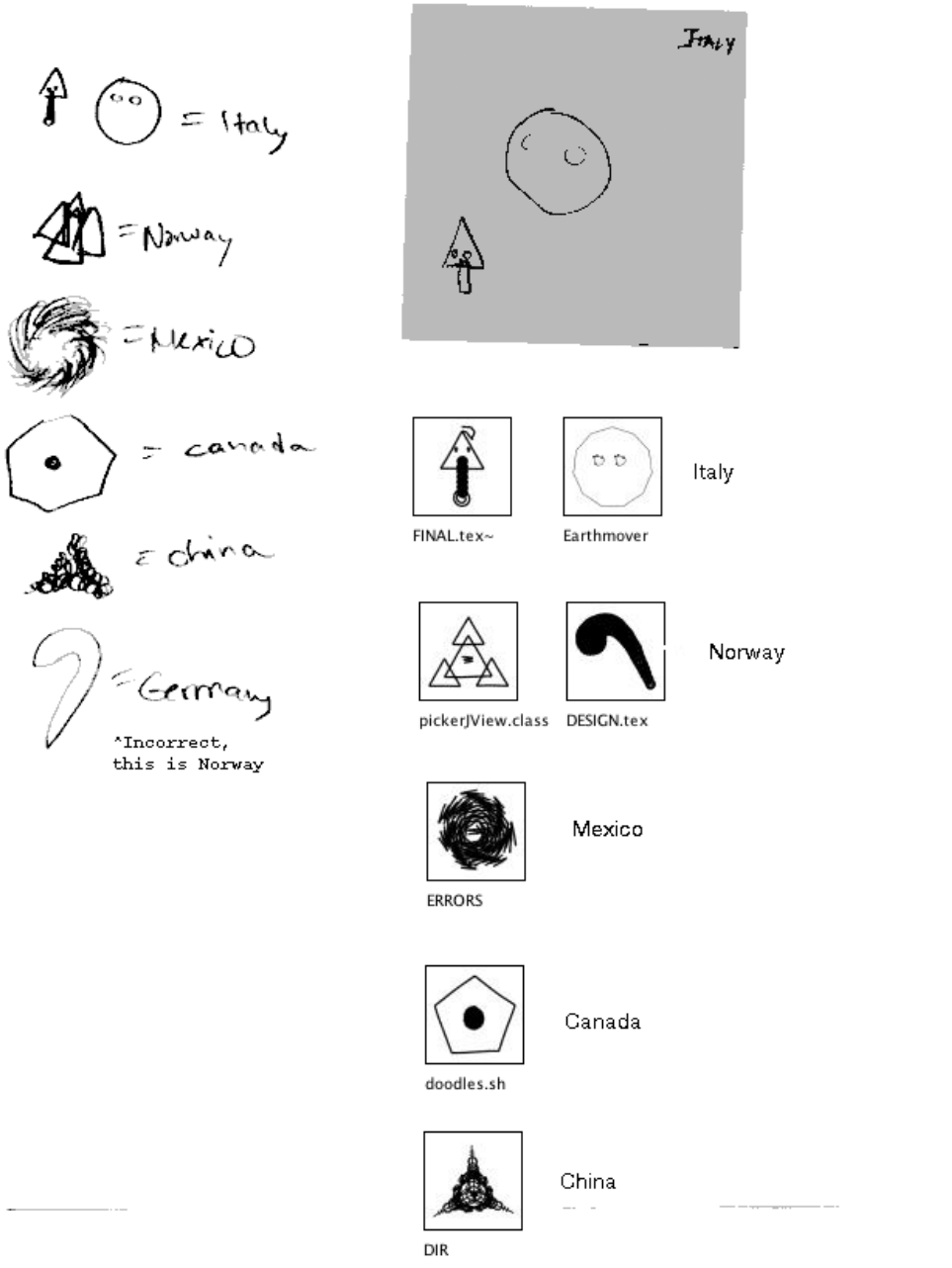
that it would be possible to learn a number of filename-content association with that amount of training did not even seem conceivable to them.

Roughly six weeks following study 2 we briefly quizzed an accessible sub-group of these people on their memory for the icons. Four people were given a printed sheet containing nine of the original icons and seven previously unseen icons; they were asked to indicate any icons they recognized. The results were remarkable: they got 4,2,2, and 1 choices incorrect, or more than 80% correct (this after only 10-20 minutes of exposure and more than a month of elapsed time).

In addition to the two studies reported in the paper, we have run a simple visual distinguishability test several times, in part to test the effectiveness of some additions to the icon grammar. People ran a program in which they viewed 100 icons for at their own pace but with instructions to spend ''a few seconds'' on each icon. They then viewed a second set of 100 icons, half of which were new. The average results appear to be in the 70% range; one group of 5 people obtained an average of 73% correct for example (chance performance=50%). This level of recognition is below that obtained for real images but nonetheless demonstrates that artificially generated icons can be distinguished and recalled with little effort. This study is not reported in the paper because it is not testing any hypothesis and because the core issue (distinguishability) is tested in more realistic ways in the other studies.



Screenshot of study 1. The url was removed for anonymous submission to SIGGRAPH.



Examples of subjects' sketches of icons from memory several days after study 2, together with the corresponding original icons for comparison.

References

BYRNE, M. 1993. Using icons to find documents: simplicity is critical. In *Proceedings of INTERCHI '93*. ACM, 446–453.

DARKEN, R., AND SIBERT, J. 1996. Navigating large virtual spaces. *Int. J. Human-Computer Interaction 8*, 1, 49–71.

INGRAM, R., AND BENFORD, S. 1995. Legibility enhancement for information visualisation. In *Proc. IEEE Visualization*, IEEE, 209–216.

PERLIN, K., AND FOX, D. 1993. Pad - an alternative approach to the computer interface. In *Proc. ACM SIGGRAPH 1993*, ACM, 57–64.

ROBERTSON, G., CZERWINSKI, M., LARSON, K., ROBBINS, D., THIEL, D., AND VAN DANTZICH, M. 1998. Data Mountain: using spatial memory for document management. In *Proc. User Interface Software and Technology (UIST)*, ACM, 153–162.

RUDDLE, R., PAYNE, S., AND JONES, D. 1997. Navigating buildings in desktop virtual environments: Experimental investigations using extended navigational experience. *J. Experimental Psychology - Applied 3*, 2, 143–159.

SEARLEMAN, A., AND HERRMANN, D. 1994. *Memory from a Broader Perspective*. McGraw Hill, New York.