

THE StageTools PACKAGE FOR CREATING GEOMETRY FOR THE WEB

DAVIDE P. CERVONE

ABSTRACT. A number of commercial software packages for generating three-dimensional graphics are available, but some useful features are missing from most of these products. The **StageTools** modules for **Geomview** try to fill this gap by including such features as interactive controls (sliders, type-in areas, check-boxes, etc.), an object-oriented approach to geometry allowing one object to be based on another (e.g., a tube around a curve), hierarchical grouped objects, objects in arbitrary dimensions with easily specified projections and transformations, and sophisticated scripting capabilities for creating electronic movie clips.

1. INTRODUCTION

The **StageTools** package is a collection of companion programs for the **Geomview** 3D graphics viewer developed at the Geometry Center, a National Science and Technology Center dedicated to producing tools and mechanisms for visualizing and presenting mathematics using contemporary technology. **Geomview** was designed simply as a viewer; it does not know how to generate the objects it shows, but relies on external programs (or *modules*), to generate the geometry it displays. The usual approach is to write a special-purpose program that creates the data needed by **Geomview** for the specific object one has in mind. This means writing a new program for each new or modified object, and requires a knowledge of **Geomview**'s various file formats. Such programming tasks are tedious, repetitive and error-prone. The first **StageTools** module, **CenterStage**, was developed as a means of simplifying this task by making it easy to generate complex, interactive objects for **Geomview** to display without having to write separate programs or know special graphics formats. The second, **StageManager**, allows these objects to be combined together to form movies that can be saved electronically for use on the web, or recorded on traditional video tape. Together, these modules form a powerful tool for communicating mathematics.

2. THE CENTERSTAGE MODULE

2.1. Creating Objects. **CenterStage** is a general-purpose program that allows the user to create geometric objects in a language that is more natural to mathematicians than most programming languages. With it, one can

generate parametric curves and surfaces, polyhedral objects, and a variety of related constructs. Objects are defined by typing a short script that specifies the important features for the object (with appropriate defaults for those that are not given explicitly). For example, a parametric surface could be given by the script

```
Function {u v} {
  let (x,y,z) = (cos u cos v, cos u sin v, sin u)
}
Domain {{-pi/2 pi/2 12} {-pi pi 24}}
```

which describes a sphere. The `Function` command specifies the parametric equations for the object, in this case, a function of the two variables u and v . The `Domain` command says that the value of u should vary from $-\pi/2$ to $\pi/2$ through 12 steps, while v should range from $-\pi$ to π in 24 steps.

Here, the function simply sets the values of x , y and z , but more complicated procedures also are possible: these could include conditional statements, loops, and so on, as a means of producing the final values of these variables. The equations used to determine (x,y,z) can include matrix and vector operations as well. For example, here is a script that represents the ruled surface between two curves:

```
Function {t u} {
  let X = (2 cos t, sin t, 0); # first curve
  let Y = (cos t, 2 sin t, 1); # second curve
  let (x,y,z) = u X + (1-u) Y; # segment between them
}
Domain {{-pi pi 24} {0 1 10}}
```

These examples are in three dimensions, but it is just as easy to produce objects in higher dimensions. For instance

```
Axes {x y z w}
Function {u v} {
  let (x,y,z,w) = (cos u, sin u, cos v, sin v)
}
Domain {{-pi pi 24} {-pi pi 24}}
```

represents a Clifford torus on the three-sphere in four-space. The `Axes` command tells how many dimensions there are and what variable is associated with each (the default is `Axes {x y z}`, so usually there is no need to give this command). In this form, the object will use `Geomview`'s n -D viewing mode; unfortunately, this mode is somewhat buggy, so `CenterStage` provides its own projection options. For instance, the command

```
Axes {x y z w} -> {x y z}
```

would perform orthographic projection into the xyz hyperplane (i.e., drop the w coordinate). The projection subspace can be defined using any of the axes, in any order. Stereographic projection also is possible. Arbitrary projections can be obtained by first rotating the object using the `Transform` command. For example,

```
Transform {XW pi/3}
```

rotates the object in the xw -plane by an angle of $\pi/3$ prior to performing the projection. The `Transform` command is quite general, and can be used to specify any number of rotations, scalings, translations, linear transformations, and user-defined functions that should be applied to the object. `Transform` can even be used to make copies of the object; e.g., a symmetric object can be created using a group action on a fundamental domain for the object (a polyhedral example appears below).

These samples give a flavor of how objects are created in `CenterStage`. Once defined, various properties of these objects can be changed using menu selections. One `CenterStage` menu controls the coloring applied to the object. For example, a surface can be colored by one of its parameters, or by one of its coordinates, or with a solid color, or even by a user-defined function. Menus also control shading characteristics (smooth or flat facets), transparency (for versions of `Geomview` that support it), presence/absence of grid lines and normal vectors, line width, and so on. See Fig. 1.

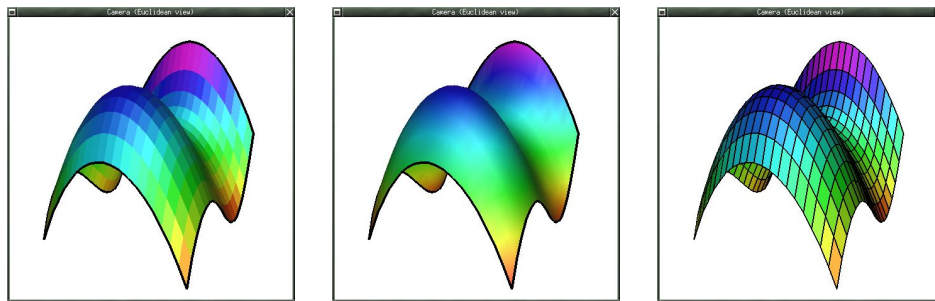


FIGURE 1. Surfaces can be made from flat facets (left), or be smoothly shaded (middle). Grid lines can be shown (right) or not.

For surface objects, another important menu selection sets the type of domain to be used. By default, `CenterStage` uses a solid patch for the range of values given in the `Domain` command, but there are a variety of other choices, including just grid lines, or various bandings, meshes, and checkerboards (see Fig. 2).

Our examples so far have all been surfaces, but `CenterStage` provides other types of objects, as well. Curves are defined in a fashion similar to surfaces. Polyhedra are described by giving a collection of vertices, and the subsets of

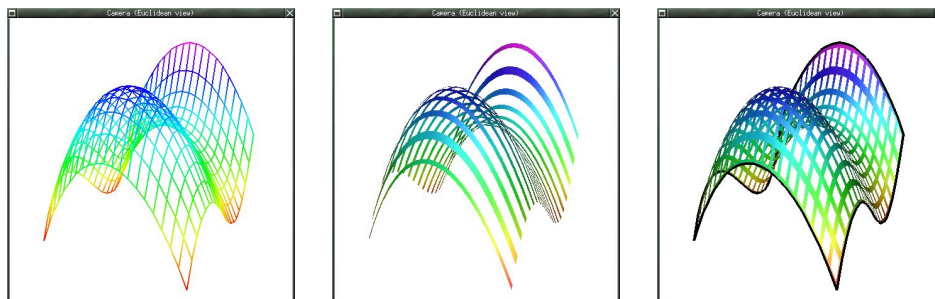


FIGURE 2. The surface domain can be solid (see figure 1), a grid (left), bands along either parameter (middle) or both (right), or a variety of other styles.

vertices that form the faces of the object (an example is given in section 2.2). Both the list of vertices and the list of faces can include formulas, or can be generated algorithmically (though this takes more programming experience).

Simple objects can be combined readily into compound objects; for example, a parametric surface can be combined with a set of coordinate axes and a slicing plane to form a demonstration of how a level set of a function of two variables relates to the graph of the function (see Fig. 3). Transformations and rotations applied to a compound object affect all the objects in the group, so a collection of objects can all be manipulated together. Values and functions defined within a group object are available to the objects within the group, so this provides a convenient means of sharing data among different objects. Very complex demonstrations can be built up in this way (the sample files that come with *CenterStage* give numerous examples of this process).

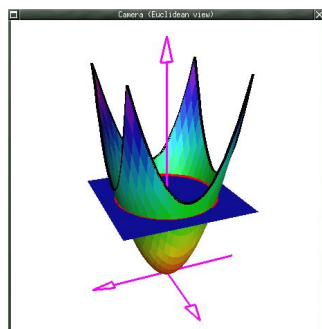


FIGURE 3. A group object contains a surface, a set of axes, a slicing plane, and the level set of the surface contained in the slicing plane. A slider can be used to control the height of the plane interactively.

2.2. Interactive Controls. One of the features that makes *CenterStage* different from most other mathematical software packages is its ability to tie interactive input devices, such as sliders, type-in areas, and check-boxes, to variables within the mathematical computations. For example, if we have a surface together with a horizontal plane slicing that surface and

the corresponding level set of the surface, the height of the plane can be controlled by a slider; the user can change the height of the plane simply by dragging the slider to a new position. Furthermore, the function being graphed could be specified in a type-in area, so that the user can change the function being displayed by typing a new one. A check-box could be used to turn off the graph of the function, leaving only the slicing plane and the level set. Such an interactive demonstration can be created in a few minutes using `CenterStage`, allowing a researcher or student to experiment with a mathematical idea by varying the parameters involved and seeing the results immediately. The ease with which changes can be made promotes exploration: the user can add or remove objects at will or change their characteristics at the touch of a button.

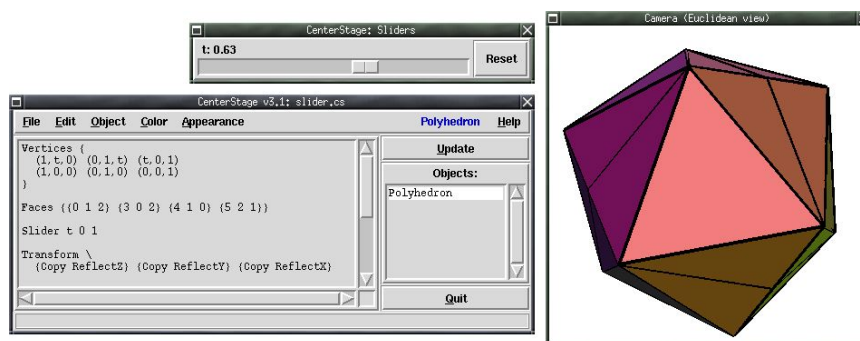


FIGURE 4. A slider can control the value of a variable. Changes in the slider cause the associated object to be recomputed automatically.

This is a simple example of a polyhedral object that is controlled by a slider (see Fig. 4):

```
Slider t 0 1
Vertices {
  (1,t,0) (0,1,t) (t,0,1)
  (1,0,0) (0,1,0) (0,0,1)
}
Faces {{0 1 2} {3 0 2} {4 1 0} {5 2 1}}
Transform {Copy ReflectZ} {Copy ReflectY} {Copy ReflectX}
```

Here, the first command defines a variable, t , whose value is given by the slider. When the slider changes, the object is recomputed and updated automatically. The value of t is used to define three of the vertices of the object, so these vertices will change position when the slider is moved. The `Transform` command at the end creates additional copies of the object: `Copy ReflectX` makes a copy with all the x coordinates negated, then `Copy ReflectY` copies both these copies with their y values negated, and finally, `Copy ReflectZ` duplicates all of these copies with negative z values. This leads to eight copies of the original triangles. When t is zero, the result is

an octahedron (three of the original triangles degenerate to lines); when t is one, the object is the cube-octahedron. Unexpectedly, for a particular value of t in between, the object is an icosahedron.

Note that experiments designed with sliders and check-boxes also have pedagogical uses. Since the user of the demonstration need not be concerned with the commands involved in making it work, he can concentrate on the mathematics rather than on programming and syntax issues, which often are a source of frustration for the inexperienced. Students presented with labwork using **CenterStage** need not learn a programming or mathematical language in order to do the assignment.

In addition to the sliders, check-boxes, and type-ins already mentioned, **CenterStage** provides pop-up menus, buttons, type-out areas (where results of computation can be displayed), and several animation devices. By default, these controls will be created in separate windows grouped by type (e.g., all the sliders will appear together), but **CenterStage** also provides custom input panels where the various input devices can be combined and placed arbitrarily; the most polished demonstrations use these.

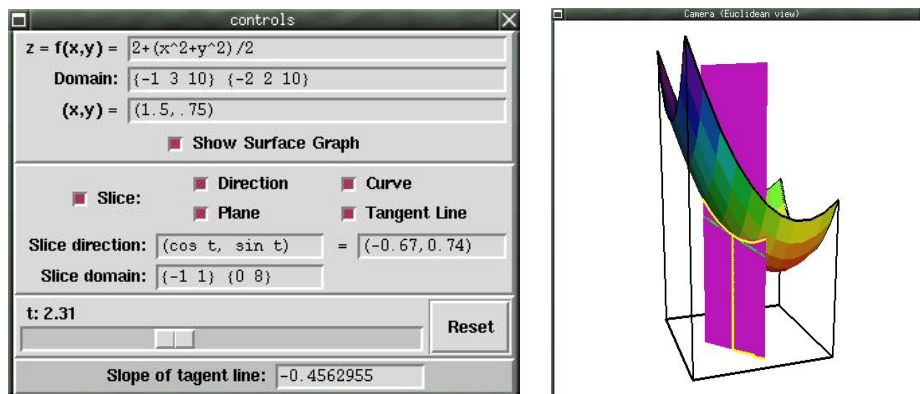


FIGURE 5. Input devices can be combined into custom panels. Here a demonstration shows how the directional derivative can be viewed as the slope of the tangent line to a curve lying on a surface: the curve of intersection of the surface with a vertical plane in the desired direction. The user can turn on and off the various elements via check-boxes, and can change the function, the point of interest, and the direction. The resulting slope is shown at the bottom.

Most of the samples that come with **CenterStage** use these input devices, often with custom panels. For example, the objects in the calculus sample files, like the one in figure 5, are designed for student use. The most complex samples provided are in the hypercube slicing demonstration in the 4D example file.

2.3. Object-Oriented Geometry. One of the more powerful aspects of the `CenterStage` program is the fact that the objects displayed are not simply treated as algebraic formulas, but as geometric entities. For example, a surface is not just an equation of two variables, but has additional structure; e.g., it knows how to compute its partial derivatives and its normal vector at any point. Other objects can use this information; so, for instance, a tangent-plane object can be created that refers to a surface object and draws a small tangent plane at a specified point. If the surface changes (say, because the user has moved a slider that controls one of its parameters) then the tangent plane will update itself automatically.

Classes of objects like surfaces and curves are fairly general; there are many ways of specifying a surface, for example. One way is parametrically by a function of two variables, but there are other natural ways to generate a surface: as a surface of revolution, or a ruled surface swept out by moving a straight line along a curve, or an offset surface produced by pushing a given surface in the direction of its normal vectors by a given distance. It is frequently difficult or tedious to compute the parametric equations for such surfaces by hand, but `CenterStage` can display them without explicit equations. For instance, there is an offset-surface object class that takes a given surface and produces the offset surface at a specified distance from the original (see Fig. 6); it does so by asking the original surface for its positions and normal vectors and performing some simple vector arithmetic. The resulting object is still a surface object, so it automatically knows how to compute its partial derivatives. The tangent-plane object described above could apply to the new offset surface even though no equations ever were given for it! This is an extremely powerful and flexible approach that more fully reflects the mathematical nature of the objects being studied than does the simple algebraic paradigm used by most graphing programs.

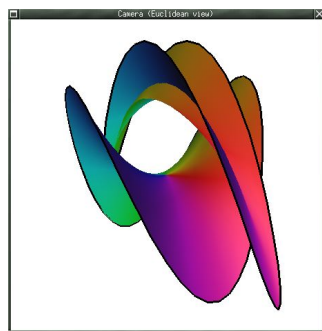


FIGURE 6. An offset surface can be linked to another surface object. The offset surface is computed without parameterizing it directly; the only value the user supplies is the distance from the original surface. In this case, the original is a saddle surface and the offset surface has self-intersection.

Another example that illustrates the power of this approach is putting a tube around a curve. One way to do this is to draw a circle in the normal-binormal plane at every point along the curve. Computing the normal and binormal vectors algebraically can be quite tedious, however, so coming up with the equations for the tube as a parametric surface can be hard. The

Tube object in **CenterStage** makes defining a tube around a curve easy: one simply specifies the shape and radius of the tube, and **CenterStage** works out the vectors automatically. The tube is still a surface object, so its tangent planes, etc., can still be obtained and you can still control its domain style and other parameters as though you had given explicit equations for it (see Fig. 7).

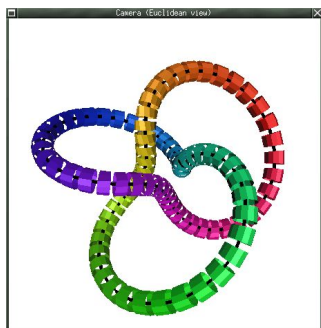


FIGURE 7. A tube can be placed around any curve object. Here we have a tube around a (3,2) torus knot. Small bands are removed from the tube so that the original curve can be seen inside.

Finally, because of this object-oriented approach, it is easy to add new classes of objects to **CenterStage**, or to make more specific versions of existing objects (e.g., the offset-surface class is a special case of the surface class). Each class knows how to generate its appropriate geometric representation, and so **CenterStage** simply has to ask the objects to draw or modify themselves at the appropriate times and pass data from one object to another as they change. The real power of **CenterStage** comes from defining useful and appropriate classes. **CenterStage** comes with a variety of surface, curve and polyhedral object classes, along with some special-purpose ones such as axes objects, vector objects, and a powerful slicing class that can cut other objects in a multitude of ways (see Fig. 8).

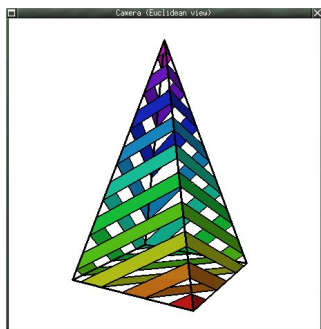


FIGURE 8. Any object can be sliced in a wide variety of ways by a slicing object. Here, a polyhedral pyramid is cut into a sequence of equally sized bands. The user does not need to specify the exact heights for the slices, but simply selects how many there will be.

3. THE STAGEMANAGER MODULE

The second module in the **StageTools** package is **StageManager**. The goal of this program is to make it easy for a researcher or educator to create

a mathematical video tape or electronic movie clip in `Geomview`. The “actors” can be created using `CenterStage`, while the “action” can be defined in `StageManager`. The basic approach is to write a “movie script” that specifies when new geometric objects appear or old ones disappear, and what transformations occur while they are in view. The movements are specified in terms of rotations, translations, scaling and other similar transformations. Pieces of compound objects can be moved independently or as part of the complete group. Several actions can be performed simultaneously, with one action being “cued” by the occurrence of some other action; e.g., one object can be asked to rotate until another one reaches a certain location.

Moreover, it is possible for `StageManager` to communicate directly with `CenterStage` in order to request it to create or modify objects as part of a movie. For example, the user might have defined a surface and an offset surface in `CenterStage`, with the offset distance given by a slider; `StageManager` could perform a loop in which it requests that `CenterStage` change the value of the slider to larger and larger distances. The result would be a movie showing the offset surface pushing away from the original (see Fig. 9).

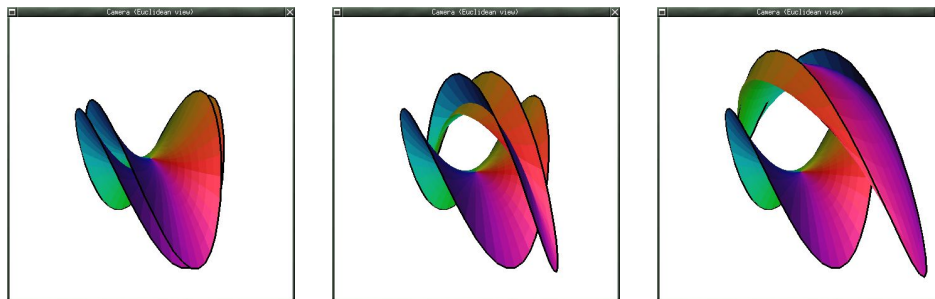


FIGURE 9. These are three frames from a movie showing an offset surface moving away from a base surface. `StageManager` automatically adjusts a `CenterStage` slider in order to generate this movie.

Such a script could be made into an MPEG movie or an animated GIF file for use on a web page in a matter of minutes. `StageManager` also could produce still images of key frames within the movie for illustrations on the web page. A longer movie can be broken down into scenes, and each can be recorded or previewed separately, so it is not hard to make changes to a script and see what the effect will be. With the proper hardware, full-length video tapes also can be generated in this way. Movies created using `StageTools` can be seen at several sites on the world-wide web; see references [3] through [11].

4. CONCLUSION

The preceding sections have tried to illustrate some of the most important features of `StageTools`, though they only hint at some of its more advanced capabilities. We close with a summing up of some of the advantages and disadvantages of using `StageTools`.

One drawback is that `Geomview` only runs under `unix`; there is no `Windows` or `MacOS` version, so `StageTools` is not available on these operating systems. On the other hand, `Geomview` does run nicely under `linux`, and there are excellent versions of `linux` for both the Macintosh and IBM PC computers.

A second problem is that, in a short-sighted decision by the National Science Foundation, the funding for Geometry Center was discontinued in 1998, so `Geomview` is no longer under development. Fortunately, however, it continues to be distributed and supported by Geometry Technologies [1], a company formed by several of the staff members from the Geometry Center. New versions are unlikely, though, so we probably are stuck with the limitations (and bugs) of the current distribution.

In terms of support, `StageTools` is not a commercial program, and is maintained by a single individual (the author) as a side project. While he makes every effort to update it regularly, it is not his primary focus, so new versions and bug fixes are not as quick in coming out as they might be for a product with a large programming staff. The current version can be obtained from the `StageTools` home page [2]. The author is aware of some bugs (and misfeatures) that may never be fully corrected.

The `StageTools` package is written in the `TCL` command language, which is an interpreted, rather than a compiled, language. This has several implications for `CenterStage` and `StageManager`. First, some of `TCL`'s syntactic peculiarities are apparent in the scripts that the user writes to define objects in `CenterStage` and movies in `StageManager`. The author has tried to minimize this, but there are still some awkward constructs, and the user has to be aware of `TCL` issues when using more sophisticated functions. Second, the object-oriented programming model employed by `CenterStage` is grafted on top of `TCL`, so there are actually two layers of interpretation going on, which makes for reduced performance. This is the result of a conscious decision (made early on) to favor flexibility over speed (though every effort was made to make things as efficient as possible within that framework).

As for advantages, the previous sections should make a number of these obvious. First, `CenterStage` makes it easy to describe objects in a reasonably mathematical language (including natural vector, matrix and complex arithmetic). Simple objects are simple to define, and more complex ones also can be handled in a reasonable way. Second, `CenterStage` and `StageManager`

handle high-dimensional data as easily as three-dimensional objects, both in terms of transformations and projections. Third, **CenterStage** provides interface-building tools (sliders, check-boxes, etc.) that make exploratory demonstrations easy to construct. Students can use these without learning a complicated programming language. Fourth, **CenterStage**'s linked-object approach makes building interrelated objects straightforward and helps preserve the connections between the various elements in a complex demonstration.

While **StageTools** does not have the generality of more mature products such as **Mathematica** and **Maple**, it does represent an important step in a new direction. As a prototype system, it serves as an example of one way to make exploratory mathematical programs more interactive. Moving away from the traditional algebra-based approach to one that is more geometric and object-oriented should provide considerable power, particularly for visualization, experimentation, and education.

REFERENCES

- [1] Geometry Technologies Home Page, [⟨URL::http://www.geomtech.com/⟩](http://www.geomtech.com/).
- [2] StageTools Home Page, [⟨URL::http://www.math.union.edu/locate/StageTools⟩](http://www.math.union.edu/locate/StageTools).
- [3] Slides for author's MTCM talk about StageTools, [⟨URL::http://www.math.union.edu/locate/Cervone/talks/00-11-23.MCTM/⟩](http://www.math.union.edu/locate/Cervone/talks/00-11-23.MCTM/).
- [4] Banchoff, T. F., and Cervone, D. P.: *Surfaces beyond the third dimension*, Providence Art Club, 31 March to 19 April, 1996; interactive electronic exhibit at [⟨URL::http://www.math.brown.edu/~banchoff/art/PAC-9603/⟩](http://www.math.brown.edu/~banchoff/art/PAC-9603/), 1997; this show was reproduced and translated into Portuguese in 2000 and is available in a new form at [⟨URL::http://alem3d.obidos.org/⟩](http://alem3d.obidos.org/).
- [5] Banchoff, T. F., Gaffney T., and McCrory, C.: Cusps of gauss mappings, [⟨URL::http://www.geom.umn.edu/locate/CVM/1998/01/cgm/⟩](http://www.geom.umn.edu/locate/CVM/1998/01/cgm/), 1998; remake of an earlier article, now including electronic movies of key examples.
- [6] Cervone, D. P., Home Page, [⟨URL::http://www.math.union.edu/locate/Cervone/⟩](http://www.math.union.edu/locate/Cervone/).
- [7] Dziadosz, S., and Hernandez, R.: *Topological zoo*, [⟨URL::http://www.geom.umn.edu/locate/zoo⟩](http://www.geom.umn.edu/locate/zoo), 1995; this project was never completed, due to the closing of the Geometry Center.
- [8] Poritz, J., and Drumm, T.: Visualizing fundamental domains, [⟨URL::http://www.georgetown.edu/faculty/poritzj/vis/intro.html⟩](http://www.georgetown.edu/faculty/poritzj/vis/intro.html), 1996.
- [9] Robles, C.: Topological zoo: hyperbolic exhibit, [⟨URL::http://www.math.ubc.ca/~robles/hyperbolic/index.html⟩](http://www.math.ubc.ca/~robles/hyperbolic/index.html), 1996.
- [10] ———, Introduction to isometries, [⟨URL::http://www.ScienceU.com/library/articles/isometries/⟩](http://www.ScienceU.com/library/articles/isometries/), 1997.
- [11] ———, Tilings and tessellations, [⟨URL::http://www.ScienceU.com/geometry/articles/tiling⟩](http://www.ScienceU.com/geometry/articles/tiling), 1998.