In Place of an Introduction*

George Francis

February 3, 2001

1 illiView

This edition of the notes has been edited for Math 428, 1997. Marginalia, such as this, provide additional guidance in anticipation of a future revision. An online distribution in the form of .dvi files is here made, 28 August 1998.

The term RTICA, short for *real-time interactive computer animator,*[1] is simply a descriptive name for what computer graphics programmers build for themselves in the course of their work. It is the harness with which they break some bronco hardware or tame some shrewish software library. It might be the workbench on which to assemble an elaborate graphics application; or it could be the graphics package itself, if that is what is destined for the marketplace. An RTICA is shared with our students and collaborators; it's merged with other software; it is abandoned and taken up again at a more propitious time when better hardware or networking becomes available.

The **illiView** collection of RTICAs grew out of my own decade long love-affair with the generations of Silicon Graphics Iris workstations. It took a new turn in 1988, when SGI graciously donated a classroom of 4D/25TGs, the Renaissance Experimental Laboratory (REL) to the National Center for Supercomputing Applications (NCSA) at the University of Illinois. Each time I teach a geometrical graphics course in the REL there is a whole new generation in the ever evolving **illiView** collection.[2] The classroom spills over into the research community at the NCSA and beyond, on the swift lines of

*The first part is paraphrased from an an article in the IEEE Computer, July, 1994, issue on visualization, edited by Arie Kaufman, titled "Interactive Methods for the Visualization of Geometry" by Andrew J. Hanson, Tamara Munzner, and George Francis. These notes were prepared for the International Summer School on Scientific and Mathematical Visualization, Ettenheim, Germany, 22-29 September 1996.

[1]Here we contracted "animation editor" to "animator," both to safeguard the acronym and prevent an unintentional latinate plural.

[2]See `http://www.math.uiuc.edu/~gfrancis`, and `http://new.math.uiuc.edu/` for current information.

TELNET, FTP and MOSAIC. In 1992, the advent of the CAVE[3] virtual reality theater accelerated this evolution. Now my students design their RTICA so that it converts easily into a CAVE application.

All 8 possible choices for C or C++, gl or OpenGL, and console or CAVE have now been developed, and reduced to minimal prototypes. But, there are reservations, limitations, and caveats. These will appear as margin notes in this edition.

What ties all these RTICAs together is an evenly balanced interpretation of all four component terms of the acronym. The typical **illiView** application is a single program of fewer than 1000 lines of basic C. It uses SGI's celebrated `gl`-library to call mathematical phenomena to life on an electronic stage. Mouse and keys generally control all conceivable parameters in concert, requiring ten-finger dexterity on the part of the pilot. To avoid breaking visual concentration, we avoid popup menus and control panels. Current parameter values and the status of options not visually obvious from the behavior on the screen can be read from an optional head-up display in the viewport.

We take the idea of interactivity further than the console. Members of **illiView** are expected to understand, modify and, occasionally, rewrite the code of their RTICA from scratch. The device this software is meant to control is the computer itself, not the VCR, the color-printer, or the slide projector. Hence our RTICAs are not particularly suitable for making frame-by-frame animations. We forego photo-realistic graphics features (anti-aliasing, subtle lighting and textures) which could paralyze all but the fastest Irises. In short, rather than treating the computer as a tool for animating a movie, we think of it as the actor to be animated by the RTICA.

There is one mathematical species that lends itself particularly well to the ministrations of **illiView**, and that is the *homotopy*. There is a continuum of sophistication involved here which extends, at one end, from the familiar Euclidean motions (reflections, translations and rotations) and their non-Euclidean generalizations, to wholly mysterious and unfamiliar phenomena, such as eversions of the sphere, at the other. The middle ground is occupied by non-rigid deformations during which the objects alter their appearance, but maintain a recognizable identity. But now, non-linear interpolations and free-form spatio-temporal splining of 2D and even 3D scenes, a.k.a. *morphing*, pushes the envelope beyond such deformations towards purely mathematical homotopies. We present some examples.

Perhaps other, newer examples are more appropriate. We'll have a look at the minimax sphere eversions, `illiVert`, the gravitational lens simulation, `graviLens`, a 3-dimensional picture of an impossible 4-dimensional object, `illUsion`, and more.

[3]This wonderful, totally immersive virtual environment includes the CAVE^{tm}, IMMERSADESK^{tm}, and INFINITY WALL^{tm}. It was created by Tom DeFanti, Dan Sandin, Carolina Cruz-Neira and Dave Pape at the Electronic Visualization Laboratory, EVL, of the University of Illinois at Chicago.

2 Homotopies

Example 1. `illiConic`
This RTICA was started by a high-school math teacher and developed for teachers' workshops for making classroom videos. `illiConic`, based entirely on Euclidean motions, is for exploring Apollonius' unification of the conic sections, Kepler's notion of continuity through infinity, and Dandelin's spheres, which touch the the nappes of the cone and the slicing place at the foci of the conic.

Example 2 `illiFive`
The parts of this RTICA were assembled in record time one summer by an **illiView** team headed by NCSA visitor, François Apéry. It includes the *cuboctahedral sphere eversion*,[4] which is a piece-wise linear realization Morin's classical eversion. This RTICA enabled us to verify the constructions, improve the parametrization, and compare it with his even more challenging algebraic sphere eversion. The motion of each vertex of this polyhedral model in 3-space was explicitly constructed by Apéry.

Example 3 `illiSnail`
This shows how to deform a Möbius band with 3 half-twists into Ulrich Brehm's *trefoil knotbox*. The mathematical significance of `illiSnail`[5] is that all of its shapes are conformal (angle-preserving) projections into flat 3-space of Blaine Lawson's ruled, minimal surfaces in the positively curved 3-sphere. These, in turn, have simple, trigonometric parameterizations with many continuous and discrete parameters. Thus its homotopies are paths in high (finite) dimensional function spaces.

Example 4 `illiBoy`
This RTICAs is representative of the tools we used to build a videotape. It is part of a long-range project to generalize to 4-dimensional space-time the methods Werner Boy invented in 1900 to construct his celebrated immersion of the real projective plane. The goal is to animate the first published sphere eversion exactly the way topologist Tony Phillips[6] sketched it the pages of Scientific American. The homotopies here are splined interpolations of surfaces initially input as 2-dimensional, hand-drawn crosssections.

---

[4]F. Apéry, Le Retournement du Cuboctaèdre, 1994 Prépublication de IRMA, Univ. Louis Pasteur, 7, rue René Descartes, 67084 Strasbourg Cedex.

[5]George Francis. The Hypergraphics Honors Seminar at Illinois. In D. Thomas, ed.. Scientific Visualization in Mathematics and Science Teaching. Assoc. Adv. Comp. in Educ., Charlottville, VA, 1995.

[6]Anthony Phillips, Turning a Sphere Inside Out, Sci. Amer., vol 214, 1966, pages 112–120.

3 Geometrical Computer Graphics

This edition contains the first ten chapters. Additional text will be handed out later. The Chap. 1–4 and 10 constitute the core course. Skip 10 if you don't have an opportunity to use the CAVE. The geometry is in Chap. 5–9. There is a complete introduction to the Unix editor `vi` in Chap. 11. We hope you will supplement these lessons with relevant chapters in *Jim Blinn's Corner* and other sources.

4 The Lecture/Demonstrations

These 4 lectures and the following 3 lessons constitute a short course given at a conference in Ettenheim, Germany, last year. In the present (1997) course we will use more and different examples, and the short-course notes will be amplified in later chapters. However, advanced students may wish to work their way through the short course at their own, accelerated, pace.

Lecture 1. **Teaching Real-time Interactive Computer Animation.**
We introduce the idea of **illiView** and how it complements other geometrical viewing packages, like *Geomview* from the Geometry Center, and Andy Hanson's *Meshview*. It deals with the way I teach several courses in Geometrical Graphics. The protagonist here is `illiConic` which was used in a 2 week teachers' workshop to produce a 75 minute videotape. I will show brief parts of this videotape during the the video-night.

Lecture 2.**A Post-Euclidean Walkabout in the CAVE**
An **illiView** team of graduate and undergraduate students from my courses put together a 10 minute piece for SIGGRAPH94. It was 'performed', perhaps 30 times for 10 visitors at a time to the CAVE wholly immersive virtual environment. I plan to describe the problems we faced working within the constraints imposed by the new technology, the nature of the audience, and subject matter. And perhaps, why this piece turned out to be so robust that it is still in the repertoire of our CAVE in Urbana.

Lecture 3.**An illiView Geodesy**
This demonstration is about visualizing ODEs, PDEs and, especially, the geodesic flow on surfaces using shared memory and distributed parallel computing. This is current work and should complement some of the other lectures given at the Summerschool.

Lecture 4. **The Minimax Sphere Eversion**
A videotape of this presentation at Supercomputing'95 will be shown during

the video night. In the hour available for the demonstration, I intend to present the new work completed since then, mainly on visualizing the double-locus surface.

5 Sample Codes and `illiShell` for 1997.

It is useful to summarize the framework within which we offer sample programs, coding prototypes, and examples written by previous students. We distinguish between 3 classes of current students: novices, apprentices and experts. Novices should work through the tutorials as presented. Their diligence will be rewarded with generous help from experts and the instructor. Experts are on their own. Apprentices are encouraged to become adventurous as soon as they feel confident. But they should also be warned that if they stray from the structure of the prototypes too radically, it will be impossible to do efficient debugging when they get into deep water.

The most efficient way for us to debug troubled modifications from a prototype is do compare code line-by-line with an SGI software tool for this purposes. Therefore, it is essential that successive modifications differ in essential places, leaving other lines untouched. [7]

5.1 Prototypes vs. Student Projects

Over the years it has proved useful to provide a set of prototype programs from which students can learn, or which students can modify to suit their purposes. In addition, current students may study and adapt programs written by previous students. The difference between these two classes of examples this. While (almost) all student programs are structural descendants from prototypes available at the time of their construction, they are not necessarily the best examples to follow. On the other hand, the small set of didactic prototypes, loosely grouped around the current illiShell, [8] change from year to year with decreasing differences. In the C/gl line, we have achieved convergence. In the more modern C++/OpenGL line, we have made a good start but there are many competing versions written by a variety of authors. In such esoterica, like JAVA/VRML/etc we have only student samples, and have made no attempt to formulate prototypes.

[7] In other words, don't ruin this debugging scheme by pretty-printing modifications. If our style offends your sensibility, wait until your program works flawlessly to pretty-print.

[8] The name, `illiShell` was chosen to signify the shell, or husk, in which to contain a more elaborate customization. It does not refer to any of the Unix shells.

5.2 How to use `illiShell97`

The current illiShell is written in C, but the graphics is written in OpenGL. Since OpenGL requires an alien windowing library, we have opted for GLUT. The CAVE library is Version 2.6. In addition, there are sample programs available using `Xlib` display. But these variants should be used only by experts who can tolerate the additional difficulties. Most of the examples older than 1997 use the C/gl/CAVE libraries that may exhibit various difficulties in compiling or even running on Irix 6.2 (the system in REL). If you are an expert, you might just reverse engineer an illiShell to see what the initial 3 lessons are about.

Please remember, however, that the essential feature of an illiShell is its ability to run in console-mode as well as in CAVE-mode. The console-mode should not be confused with running a CAVE-worthy program in simulator mode on a console. The console-mode should be compile also without the CAVE libraries present. This feature should be maintained in all modifications.

5.3 Tutorials leading to `illiShell97`

There are a series of elementary RTICAs, with nicknames like `oc1.c, tr1.c, skel.c` etc., which accompany the introductory lessons. These do not involve CAVE code and therefore use C/gl rather than OpenGL to avoid extraneous windowing baggage. However, if you wish to start with OpenGL from the start (which you must in order to compile your code on an WiNTel or other, non-SGI platform) you can use the relevant versions of the same programs. But we strongly recommend that you do not mix gl with OpenGL.

Experts are welcome to work in C++, or even in JAVA. But if you do, you will be required to explain deviations from the C prototypes to the instructor and to the class. These explanations will help you clarify your own thinking, as well as helping your classmates understand the purpose of these contemporary styles.

Apprentice programmers who already know C but are inclined to learn C++ in the course, should consider translating the elementary RTICAs into C++ not just superficially, but conceptually, without abandoning their essential, pedagogical nature.