

7. Diagrams, videos, and virtual reality

7.1. Diagrams with *pic* and *Mathematica*

Geometrical text without line drawings is unthinkable. Even Euclid used line drawings and, it is believed by historians, that some conventional illustrations were carefully copied, first by hand, later by print media from generation to generation from his originals. Mathematical drawings usually have a precise, if complicated, formula driven description. Therefore, it is logical to expect that modern typographical instruments, like postscript printers, could be manipulated by the mathematical author. Unfortunately, this is not the case. An author using a scientific word processor, such as T_EX can write text and see the output's form almost simultaneously. A postscript file has such a detailed prescription that directly editing it is impractical. What is wanted is a means for effective interaction between author, code and produced picture. There are tools for doing this, which themselves are more or less difficult to learn and to use. *Mathematica* was our first choice because most mathematicians already know enough *Mathematica* to master the drawing routines. In fact, most diagrams in this work were initially designed and coded in *Mathematica*. But *Mathematica* was not specifically designed for this task, and the products were generally disappointing from a graphical viewpoint. To bring them up to snuff required more expertise than warranted. Therefore, we were overjoyed when it proved that *pic* was not only perfectly up to the task, but provided graphically superior results. We should point out that *pic* does not allow the use of colors in the graphics. This becomes an issue if one is planning to present illustrations over the internet.

As an example, we present the code for producing a simple figure, first in *Mathematica*

and then in *pic*. A rectangle is drawn in the first quadrant with two vectors based at a single point.

Mathematica code for the illustration below

```
Show[
Graphics[
{Line[{{0,0},{1.7,0},{1.7,1},{0,1},{0,0}}],
{Thickness[.02],Line[{{v0x,v0y},{v0x+.9,v0y}}],
Line[{{v0x,v0y},{v0x+.4,v0y+.5}}]},
arrowHead[{{v0x,v0y},{v0x+.9,v0y}],arrowHead[{{v0x,v0y},{v0x+.4,v0y+.5}}]},
Axes ->True,
Ticks ->None,
PlotRange ->{-.2,1.9,-.2,1.2}
]
];
```

pic code for the illustration below

```
.PS 2
v0x=.4;v0y=.2;
line from 0,0 to 1.7,0;line from 1.7,0 to 1.7,1;
line from 1.7,1 to 0,1;line from 0,1 to 0,0;
line -> thick 1.25 from v0x, v0y to v0x+.9, v0y
line -> thick 1.25 from v0x, v0y to v0x+.4, v0y+.5
"$ a $" at v0x+.1, v0y+.25
"$ b $" at v0x+.4, v0y -.06
```

```
"$ v_0 $" at v0x -.05, v0y -.05
```

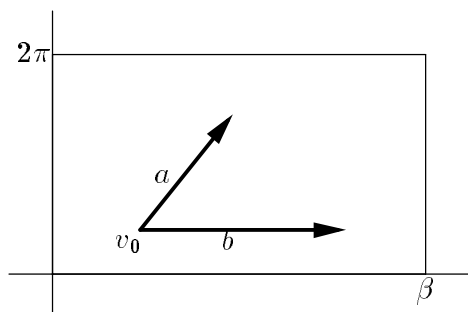
```
axes(-.2, 1.9, -.2, 1.2);
```

```
"$ \beta $" at 1.7, -.08
```

```
"$ 2\pi $" at -.08, 1;
```

```
.PE
```

```
box graph
```



7.1.1. *pic*: a language for embedding diagrams in \TeX documents

The syntax of the *pic* language is extremely simple; there are very few available commands and functions. In this way, it resembles *C* programming language. The resemblance is not mere coincidence. Both languages were developed at AT&T Bell Laboratories. Like the *C* programming language, *pic* supports the definition and use of a limitless variety of functions. For example, we defined functions for linear fractional transformations acting on \mathbb{C} and the Poincaré extension of Möbius transformations²⁰ acting on \mathbb{R}^3 . Many of the illustrations in Chapter 3 were produced using compositions of functions that include these.

The code for each illustration can be written in its own file and called from within a \TeX document. The length of a \TeX document is limited. This limit can be exceeded by

²⁰We wish to thank Bill Abikoff for the formulae used here for the Poincaré extension of Möbius transformations.

illustrations with extremely long prescriptions. We have been able to avoid reaching this limit by relatively minor steps at reducing the number of lines in drawing curves. On the other hand, there seems no limit to the number of computations that the *pic* processing software, *gpic*, will perform. The compilation time for producing the postscript files can be quite long for illustrations with many points computed with several composed functions. For example, the illustrations of Steiner nets in Chapter 5 each took approximately 30 seconds to compile. The entire set of illustrations in Chapter 5 requires about five minutes to compile on a *Sun Sparcstation 5* using *gpic*.

Below are a few simple functions²¹ defined for use within *pic*. The first group yield the real and imaginary parts that result from the binary operations of complex arithmetic. Those familiar with *C* programming or with macro definitions in \TeX will recognize the $\$1$ as indicating the first parameter in the function call, $\$2$ the second parameter, and so on.

pic **code for binary operations in complex arithmetic.**

```
define addre {($1)+($3)}
define addim {($2)+($4)}
define subre {($1)-($3)}
define subim {($2)-($4)}
define mulre {($1)*($3)-($2)*($4)}
define mulim {($1)*($4)+($2)*($3)}
define divre {((($1)*($3)+($2)*($4)))/((($3)*($3)+($4)*($4))}
```

²¹A defined *function* in *pic* is actually a replacement macro, that is, wherever the function name and parameters are found in the *pic* code, they are replaced with the code in the written definition and the parameter values are inserted at the positions indicated in the definition.

```
define divim {(-(1)*($4)+($2)*($3))/((3)*($3)+($4)*($4))}
```

The above definitions are used to define computations for a linear fractional transformation, $M(z) = \frac{az + b}{cz + d}$, $z \in \mathbb{C}$, and $a = \text{aare} + i\text{aaim}$, $b = \text{bbre} + i\text{bbim}$, etc.

A linear fractional transformation in *pic*.

```
define Lftre {divre(
addre(mulre((1),($2),aare,aaim),mulim((1),($2),aare,aaim),bbre,bbim),
addim(mulre((1),($2),aare,aaim),mulim((1),($2),aare,aaim),bbre,bbim),
addre(mulre((1),($2),ccre,ccim),mulim((1),($2),ccre,ccim),ddre,ddim),
addim(mulre((1),($2),ccre,ccim),mulim((1),($2),ccre,ccim),ddre,ddim))}
define Lftim {divim(
addre(mulre((1),($2),aare,aaim),mulim((1),($2),aare,aaim),bbre,bbim),
addim(mulre((1),($2),aare,aaim),mulim((1),($2),aare,aaim),bbre,bbim),
addre(mulre((1),($2),ccre,ccim),mulim((1),($2),ccre,ccim),ddre,ddim),
addim(mulre((1),($2),ccre,ccim),mulim((1),($2),ccre,ccim),ddre,ddim))}
```

The programmer must specify the values for *aare*, *aaim*, etc, prior to calling these functions. The same basic arithmetic functions are used to define the coordinate functions for the Poincaré extension of Möbius transformations.

The Poincaré extension of a Möbius transformation in *pic*.

```
define poincareExtensionX (((aare*$1-aaim*$2+bbre)*(ccre*$1-ccim*$2+ddre)+
(aare*$2+aaim*$1+bbim)*(ccre*$2+ccim*$1+ddim) +
(aare*ccre+aaim*ccim)*$3*$3)/
((ccre*$1-ccim*$2+ddre)*(ccre*$1-ccim*$2+ddre)+
(ccre*$2+ccim*$1+ddim)*(ccre*$2+ccim*$1+ddim)+
```

```

(ccre*ccre+ccim*ccim)*$3))

define poincareExtensionY (((aare*$2+aaim*$1+bbim)*(ccre*$1-ccim*$2+ddre)-
(aare*$1-aaim*$2+bbre)*(ccre*$2+ccim*$1+ddim) +
(aaim*ccre-aare*ccim)*$3*$3)/
(((ccre*$1-ccim*$2+ddre)*(ccre*$1-ccim*$2+ddre)+
(ccre*$2+ccim*$1+ddim)*(ccre*$2+ccim*$1+ddim)+
(ccre*ccre+ccim*ccim)*$3))

define poincareExtensionZ (sqrt((mulre(aare,aaim,ddre,ddim)-
mulre(bbre,bbim,ccre,ccim))*(mulre(aare,aaim,ddre,ddim)-
mulre(bbre,bbim,ccre,ccim)))+(mulim(aare,aaim,ddre,ddim)-
mulim(bbre,bbim,ccre,ccim))*(mulim(aare,aaim,ddre,ddim)-
mulim(bbre,bbim,ccre,ccim))))*$3/
(((ccre*$1-ccim*$2+ddre)*(ccre*$1-ccim*$2+ddre)+
(ccre*$2+ccim*$1+ddim)*(ccre*$2+ccim*$1+ddim)+
(ccre*ccre+ccim*ccim)*$3))

```

As far as we know, we are the first to make use of such function capabilities for graphics in *pic* using complex mappings. We have also written code for other familiar transformations, such as stereographic projection and rotations in \mathbb{R}^3 , which were used for illustrations in this work.

Many of the caveats that apply for those programming with the *C* programming language also apply to *pic* programmers. One of the advantages most appreciated by us in using *pic* was the ease with which one can insert labels within an illustration. *pic* allows the use of all available \TeX fonts and symbols.

7.2. Videos

For each of the past two years a ten minute video tape was produced for presentation at the Winter Joint Mathematics Meetings of the AMS, MAA, ASL, AWM, and NAM. In this chapter we present a description of each video.

7.2.1. Animating Teichmüller Spaces and Kleinian Groups

Text for a presentation at the Joint Mathematics Meetings

in Orlando, 1/13/96

In the early moments of this video, while I introduce the story of Teichmüller space, you will be seeing a canonical tiling of the unit disc by a deck transformation group acting on a fundamental polygon. We will see it twice for good measure.

The Teichmüller space of closed genus two Riemann surfaces is known to be diffeomorphic to an open cell in R^6 . Each point in this open cell corresponds to a surface with a certain set of markings. The markings lead to parameters that can be made intrinsic in that they appear as the measure of uniquely determined arcs and angles in a canonical hyperbolic polygon serving as a model for the marked Riemann surface.

What kind of hyperbolic polygon can actually be a Riemann surface? There are just two conditions that need be met. The sides must be congruent in pairs. It is these pairs that are glued together to form the surface from the polygon. The second condition is that all vertices of the polygon that come together on the surface in this gluing process must have angles that sum to 2π (or an integral divisor of 2π , if we allow ramification points.) Elements of the deck transformation group map the polygon onto isometric copies. Images of the polygon under the group's action cover a neighborhood of the vertex with no overlapping.

Why is it sufficient to confine our investigation to *hyperbolic polygons in the unit disc* when there certainly are *abstract* Riemann surfaces? An abstract Riemann surface lifts, along with its conformal structure, to its universal covering space. If the genus of the surface is greater than one, then its universal covering space is conformally equivalent to the unit disc. The deck transformation group of the surface is pushed forward from the abstract universal covering space to act on the unit disc and there it exists as a Fuchsian group of the first kind, acting freely on the unit disc and isomorphic to the surface's fundamental group. The orbit space in the unit disc is itself a Riemann surface, equivalent to the original abstract surface with its given conformal structure.

One can envision a topological embedding of a closed Riemann surface of genus two into R^3 as the wrapping up of an eight sided hyperbolic polygon.

To say that a given surface is represented by an infinite number of distinct points in its Teichmüller space is the same (up to homotopy equivalence) as saying that a polygon representing a surface can be cut and pasted in an infinite number of ways and still represent the same surface.

The markings of a surface can be thought of as those cuts made to enable the surface to lie flat in the hyperbolic plane. There the cut-lines correspond to pairs of congruent sides of a polygon. Thus the sides of the polygon project to loops on the surface and there generate the surface's fundamental group. In the unit disc, the side identification transformations generate the deck transformation group. A new set of markings results in a distinct new point in the Teichmüller space.

We start here with a polygon whose *opposite* sides are identified. We know that any topological plane figure modeling a surface can be cut and pasted so that all four sides

corresponding to each handle grouped together, that is, in the canonical order $ABA^{-1}B^{-1}$.

We begin with such a polygon when we construct the canonical polygon that yields intrinsic moduli for the Teichmüller space. Each side identifying transformation has a unique stabilized geodesic associated with it known as its axis. It is along this arc that we wish to cut to construct a side of the canonical polygon. The lengths of the sides that lie along the axes will be the same as those on the original polygon, however their new positions yield uniquely determined angles that provide three of the six real parameters of the Teichmüller space. The canonical polygon not only retains the original conformal structure, its new markings lie in the same homotopy equivalence classes as the original ones. Thus the original and the canonical polygons represent the same point in the Teichmüller space $T(2;0)$. This construction was suggested by Linda Keen in a striking and beautifully geometric article that appeared in the *Annals of Mathematics*.

As we lengthen one pair of sides, for the polygon to remain a surface model, the other pair of sides associated with the handle shrinks. At the limit, its length shrinks to zero and the polygon is deformed to one that represents a double torus with one of its “tubes” pinched closed or, equivalently, a single torus with two points removed. This surface represents a point on the *boundary* of the Teichmüller space of closed genus two surfaces.

Here we present a profound analogy. An arc in a universal covering space connecting two congruent points will project to a closed loop in the quotient space. In the context of a closed genus two Riemann surface, the unit disc is the universal covering space. An arc connecting two points congruent with respect to the action of the deck transformation group will project to a closed loop in the Riemann surface, represented by the central polygon. in the context of the Teichmüller space, the Teichmüller space itself is simply

connected and will be the universal covering space here. The Mapping class group acts discretely on the Teichmüller space and this action produces a quotient space, called the Riemann space or *moduli* space. The Riemann space is comprised of all surfaces with conformally distinct structures; the distinction of different markings is erased by the action of the mapping class group. A one parameter deformation of surfaces which begins and ends at two surfaces that are conformally equivalent but which have two distinct *markings* will be an open curve in the Teichmüller space, but will project to a closed loop in the Riemann space of closed genus two surfaces. Of course these curves exist in R^6 .

We now turn our attention to the three dimensional case of hyperbolic space. One can extend the action of the group of two dimensional Möbius transformations which acts on the complex plane, to conformal transformations of the upper half-space.

We present here one example of each possible commutative Kleinian group. Their importance for us is two-fold. First, they are the most simple in structure and, therefore, a useful tool in introducing Kleinian groups. Second, finite extensions of these constitute the notable exceptions in many theorems in the field. We focus on the one and two dimensional manifolds that are stabilized, that is, mapped onto themselves by the action of these Abelian groups.

1** The first example is of an infinite cyclic group generated by a loxodromic Möbius transformation. All of the iterated images of a point on the boundary plane lie on a curve like the flat spiral shown here. The axis of the transformation is a semi-circle which is orthogonal to the boundary plane. The locus of all points equidistant from the axis in the hyperbolic metric, is a surface stabilized under the group's action. A typical example is shown here as a banded surface. A spiral curve lying on this surface is a typical example

of a stabilized one-manifold lying within hyperbolic 3-space.

2** An elliptic Möbius transformation is, essentially, a rotation of the space about the transformation's axis. Once again, each of the surfaces that is equidistant from the axis is mapped onto itself. In fact, any surface that is symmetric with respect to this axis is stabilized. A finite cyclic group is generated by an elliptic Möbius transformation of finite order.

3** Two elliptic transformations of order two that have a common, finite fixed point generate a Klein-4 group. Each transformation causes the end points of the other's axis to be interchanged. Here we may see that a right circular cone is the surface equidistant in the hyperbolic metric from a Euclidean straight line orthogonal to the boundary plane.

4** An elliptic transformation of finite order along with a loxodromic transformation that shares the same axis, together generate an Abelian Kleinian group that is the direct sum of a finite cyclic group with that of an infinite cyclic group.

5** Parabolic transformations are simply Euclidean translations when the single fixed point is the complex point at ∞ . In order to form a commutative group, all parabolic generators must share the same fixed point. In this context we know that the two possible commutative groups are either an infinite cyclic or a free group on two generators.

The video ends with a picture of the fundamental polyhedron of an infinite cyclic group generated by a single loxodromic transformation with trace having modulus less than two.

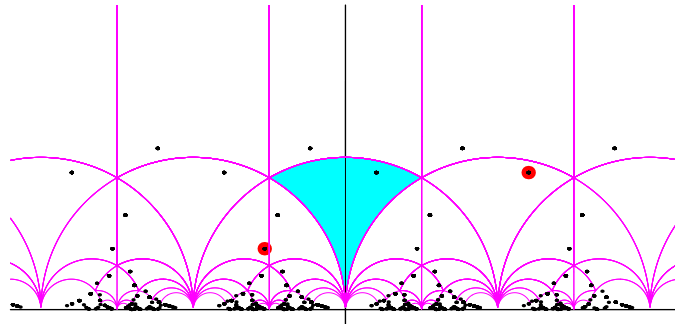
7.2.2. Conformal Embeddings of Tori in \mathbb{R}^3

**Text for a presentation at the Joint Mathematics Meetings
in San Diego, 1/8/97**

You will be asked to divide your attention between the overhead slides and the video screen.

Garsia showed us in 1961 that it is possible to conformally embed any closed Riemann surface in R^3 . Still there are only isolated examples of such embeddings even in the simplest of cases. Here we discuss the case of closed genus one, that is, topological tori conformally equivalent to parallelograms.

The Teichmüller Space $T(1,0)$



Orbit of τ under action of $PSL(2, \mathbb{Z})$.

To set the context, we consider the Teichmüller space $T(1,0)$, which we view as the upper half-plane. Each point τ in the upper half-plane represents a torus together with its conformal structure. Further, each τ belongs to an equivalence class which is the orbit of itself under the action of $PSL(2, \mathbb{Z})$. Here we consider $PSL(2, \mathbb{Z})$ as an isometry group of the hyperbolic plane. To navigate a path through this Teichmüller space, we continuously vary the parameter τ . For each τ we wish to construct a torus that holds

the same conformal structure as the parallelograms associated with that parameter value. With these constructions, we can watch the tori deform on our screen as we fly through the Teichmüller space of conformal tori.

We already have constructed conformal maps from all rectangles to tori and, thus, from all parallelograms with parameter τ that lie in equivalence classes with pure imaginary numbers.

The parameter lines on this torus intersect at equal angles. Whenever this is the case on a topological torus, we say that the torus is *preconformal*. We now show a composition map from the rectangle to the torus that is conformal in its limit. We begin by cutting the rectangle into n vertical strips. Each strip we translate to center its bottom edge at the origin. The rectangle is normalized to have height 2π , so the exponential function gives us an annulus. A Möbius transformation that preserves the unit disc is followed by stereographic projection which yields a wedge shaped section of a sphere.

If we applied the appropriate Möbius transformation, the dihedral angle of our sphere wedge will allow us to glue it together with all the pieces coming from the vertical strips so that there is no gap and no overlapping. Off the gluing seams, the map is clearly conformal; it is the composition of conformal maps. Unfortunately, these seams, where conformality fails, are dense in the limit, when we allow the number of vertical strips to go to infinity. That this limit map is conformal is the subject of a preprint available on my web homepage?

We will construct topological tori from core curves. For rectangles the core curves can be circles, as we see here; for general parallelograms, the core curves may not be planar. The grid of parameter lines will be constructed from adapted frames on the curves.

Frenet and Bishop Frames

$$\text{Let } V = \begin{pmatrix} T \\ N \\ B \end{pmatrix}$$

$$A_B = \begin{vmatrix} 0 & k_1 & k_2 \\ -k_1 & 0 & 0 \\ -k_2 & 0 & 0 \end{vmatrix}$$

$$A_F = \begin{vmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{vmatrix}$$

Then $V' = A_i V$, with $i = B, F$.

Two space curves are shown here with ribbons attached to indicate the orientation of the adapted frames. The sphere is attached to what we refer to as a Bishop curve, after Richard Bishop who introduced the framing in 1975 as an alternative to the classic Frenet framing.

A red square is in the upper right-hand corner of the screen. This is the unit square in the first quadrant of the frame's parameter space. The thick yellow horizontal line is a normal development curve in the frame parameter space. We use the term *normal development* to refer to a frame's parameter curve in order to distinguish it from the parameter lines we refer to on the tori. As the frames fly along the space curves, we note a marked difference in the twisting of the two frames. The ribbons attached to the curves

help highlight this difference. In general we find the Frenet frame twists much more than does the Bishop frame. This is because the Bishop frame follows relatively parallel fields so that the frame elements twist just enough to keep themselves orthogonal.

The normal development, or frame parameters, uniquely determine a space curve. The adapted frame is determined up to a Euclidean rotation for the Bishop frame. Both frame and space curve are determined by one of the two ordinary differential equations shown on the slide. We use a fourth order Runge-Kutta method to integrate the development and obtain the curve and adapted frame.

We focus our attention on the Bishop curve and watch its shape change as we move the corresponding normal development in the parameter space. We restrict these normal developments to lie on straight line segments. The normal development determines the shape of the space curve and the orientation of the adapted frame.

We view both Frenet and Bishop curves together once again. These curves have both been generated by the same straight line segment viewed first as lying in the $\tau\kappa$ -plane for the Frenet frame and then as lying in the Bishop frame's parameter space.

As we construct core curves for our topological tori, we wish to have control over the shape of the space curve and over the twisting of the frame. Frenet framings have proven quite awkward in shaping the space curves. Further, the excessive twisting raises issues of computational accuracy.

Bishop Frame

I. For a normal development curve that lies on a straight line segment in the k_1k_2 -plane,

$$fk_1 + gk_2 = 1$$

and the osculating sphere radius,

$$r = \frac{1}{\sqrt{f^2 + g^2}}.$$

II. Let D =normal development curve

in the k_1k_2 -plane

$$J : D \rightarrow \gamma$$

E_2 =rotation in k_1k_2 -plane by α .

E_3 =rotation by α in R^3 fixing $\gamma(0)$

and preserving $N(0)B(0)$ -plane

$$\text{Conjugation rule } J = E_3^{-1} \circ J \circ E_2 .$$

We remark here on two interesting features of the Bishop frame. First, when the normal development lies on a straight line segment, then the resulting space curve lies on a sphere, called the *osculating* sphere.

The radius of this sphere is the reciprocal of the distance that the normal development lies from the origin in the parameter space. As the normal development curve is moved closer or further from the parameter space origin, the osculating sphere changes size. No matter what the orientation of the normal development, the space curve will remain on the sphere.

To introduce the second feature of the Bishop frame, we view the space curve as the image of the normal development curve. The mapping is the integration process and we denote it by the letter J . The feature of interest is that a rotation of the normal development in the frame parameter space results in a Euclidean rotation of the generated space curve

by an equal angle in R^3 , that is, the map J obeys a conjugation rule. This remarkable relationship gives us a great deal of flexibility when we glue segments together. The Frenet frame's normal and binormal vectors must have a fixed orientation with respect to the curve's acceleration vector. On the other hand, the Bishop frame can be reoriented with respect to the curve's acceleration vector without changing the curve's shape.

As we construct closed curves, we glue together space curve segments which are generated from independent normal developments. We note that a necessary condition for conformality in a map from a parallelogram to a torus is that there exist a grid of parameter lines on the torus which has intersection angles all of which are equal. As mentioned earlier, we say such a torus is *preconformal*. We can color the surface of tube segments to indicate how closely a grid on it approaches the condition of preconformality. Further, for the Bishop frame, we are allowed to rotate glued segments by an arbitrary angle about their common tangent. This flexibility in gluing together segments and their surrounding tubes is definitely not available with the Frenet frame.

We use a spectrum of colors to indicate how close the measures are of the angles at which parameter lines intersect each other on the tubular surfaces. We use red and dark blue coloring to indicate where on the surface the intersection angles are respectively larger or smaller than a specified range. The angles that lie within the range are assigned colors in the spectrum between green and cyan. We are using this color map to identify factors that produce preconformal structures and are searching for a way to modify the preconformal structures to obtain conformality, as we did in the case for rectangles.

Here ends the presentation on the search for conformal tori.

7.3. Totally immersive 3-D environment (CAVE™)

7.3.1. Introduction.

As we develop RTICA, the CAVE provides an arena for presentation and interaction with experts in numerous fields. For example, many of the guest presenters for the *Mathematics in Science and Society* series, which was sponsored by UIUC's Department of Mathematics, made visits to the CAVE during their stay on campus. Receiving observations and questions from individuals with a broad range of background, experience, and expertise provides invaluable insights, points us in new and fruitful directions and leaves us with renewed enthusiasm and inspiration to investigate and explore. Following such interactions, we have produced visualizations to reveal the observations made by visitors and to illuminate features that surround questions posed by the distinguished visitors.

Most of the information in the following two sections was obtained from the web pages of the Electronic Visualization Laboratory (EVL) at University of Illinois at Chicago. In particular, all of the related pages are linked to the listings at <http://www.evl.uic.edu/pape/CAVE/prog/CAVEGuide.html>.

7.3.2. CAVE description.

The CAVE is projection-based virtual reality (VR) system with screens that surround the field of vision and with speakers that surround the room. 3-D computer graphic images are projected onto three walls and the floor of a 10'x10'x9' cube, completely filling the visual range of the viewer. Head and hand tracking systems produce the correct stereo perspective and isolate the position and orientation of a 3-D input device. A sound system provides audio feedback. A three-button *wand* (functioning like a 3-D mouse) provides direct,

intuitive interaction with the animation program. Lightweight stereo glasses equipped with a head tracking device allow the viewer to walk around within the CAVE cube. With this interaction with the animator, the viewer can investigate by walking around and even through the virtual objects in the animations.

Multiple viewers can share virtual experiences and easily carry on discussions inside the CAVE. One user is the active viewer/navigator, controlling the stereo projection reference point, while the rest of the users are not tracked and do not have the correct perspectives drawn for where they stand in the CAVE cube.

The CAVE was designed by the Electronic Visualization Laboratory (EVL) at UIC and premiered at the ACM SIGGRAPH 92 conference. Electrohome Marquis 8000 projectors throw full-color workstation fields (1024x768 stereo) at 96 Hz onto the screens, giving approximately 2,000 linear pixel resolution to the surrounding composite image. Computer-controlled audio provides sound from multiple, surrounding speakers. A user's head and hand are tracked with Ascension tethered electromagnetic sensors. Stereographics' LCD stereo shutter glasses are used to separate the alternate images going to the eyes. The stereo emitters are little white boxes placed around the edges of the CAVE. They are devices that synchronize the stereo glasses to the screen update rate of 120Hz or 96Hz. A Silicon Graphics Onyx with three Reality Engines is used to create the imagery that is projected onto the walls and floor. A wand (3-D mouse) with buttons is the interactive input device. The wand has three buttons and a pressure-sensitive joystick. It is connected to the CAVE through a PC which is attached to one of the Onyx's serial ports. A server program on the PC reads data from the buttons and joystick and passes them on to the Onyx. The wand has a sensor at the front tip that returns its (x, y, z) position inside the

CAVE cube to the application program. The sensor also returns three orientation angles: azimuth, elevation, and roll. Thus, an animation CAVE program can allow a user to control actions in the animation by wrist motions as well as button pushes. The CAVE's theater area sits in a 30x20x13-foot light-tight room, provided that the projectors' optics are folded by mirrors. The projectors and the mirrors for the side walls are located behind each wall. The projector for the floor is suspended from the ceiling of the CAVE.

CAVE is both a recursive acronym (Cave Automatic Virtual Environment) and a reference to "The Simile of the Cave" found in Plato's Republic, in which the philosopher explores the ideas of perception, reality, and illusion. Plato used the analogy of a person facing the back of a cave alive with shadows that are the only basis for ideas of what real objects are.

7.3.3. Available function libraries

The CAVE Library has all functions required in a CAVE program. Examples of such functions are synchronization of CAVE devices, synchronization of the walls, calculations of the stereo transformations, and many other CAVE-specific tasks. The CAVE Simulator is a library that runs on any workstation which supports SGI's IrisGL library. The simulator allows a programmer to do preliminary CAVE application development, place objects in a scene, or choose rendering without having to use the actual CAVE hardware. For every frame of an animation, two views are produced, one for the left eye and one for the right eye. The CAVE library display routine will call the drawing routine twice for every frame.

In the previous version of the CAVE, up to five SGI machines were connected via a ScramNet reflective memory. Four of the machines were dedicated to running applications

for each of the four CAVE walls, and an additional machine was used to synchronize the other four. Under this scheme, four separate copies of a user's application were running, one per machine. Applications ran in serial mode; at each frame new data was calculated and then the scene was rendered for display in the CAVE.

The new version of the CAVE runs on a multiprocessor SGI Onyx; this one machine performs all the CAVE tasks. When an application is run, it forks into several concurrent processes. There is an independent rendering process running for each wall, plus a computations process to perform calculations. With this system, some method is needed to communicate changes in data between the applications and rendering processes; normally forked processes all have separate copies of a program's data, which means that changes made by the computations process would not be seen by the rendering process.

By putting the data in shared memory, computations can be performed in parallel with the rendering, with all processes using the same data. the CAVE library provides a utility function to create a shared memory arena, from which one can allocate shared data.