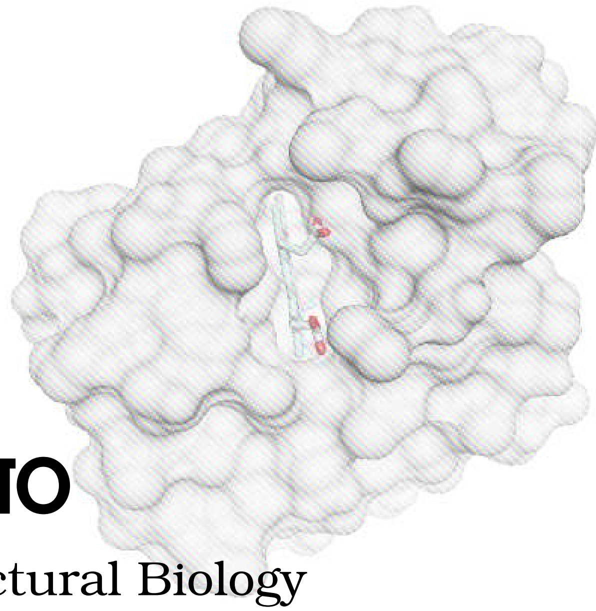
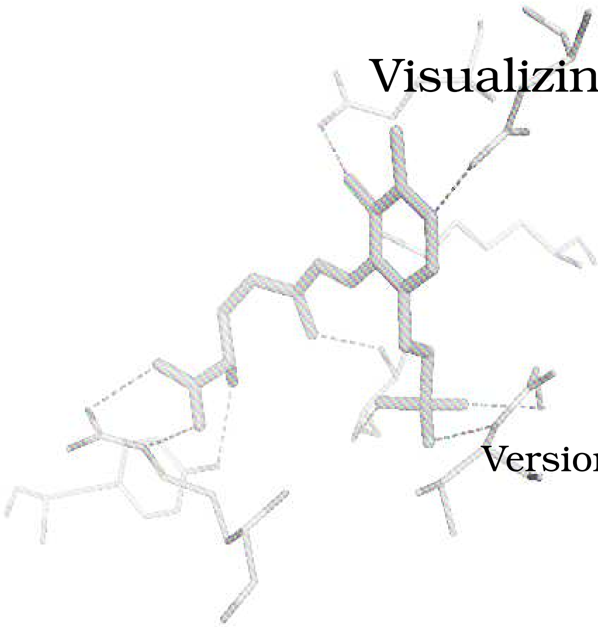


DINO

Visualizing Structural Biology
v0.8



User Manual
Version February 29 2000

© 1998/1999/2000 Ansgar Philippsen

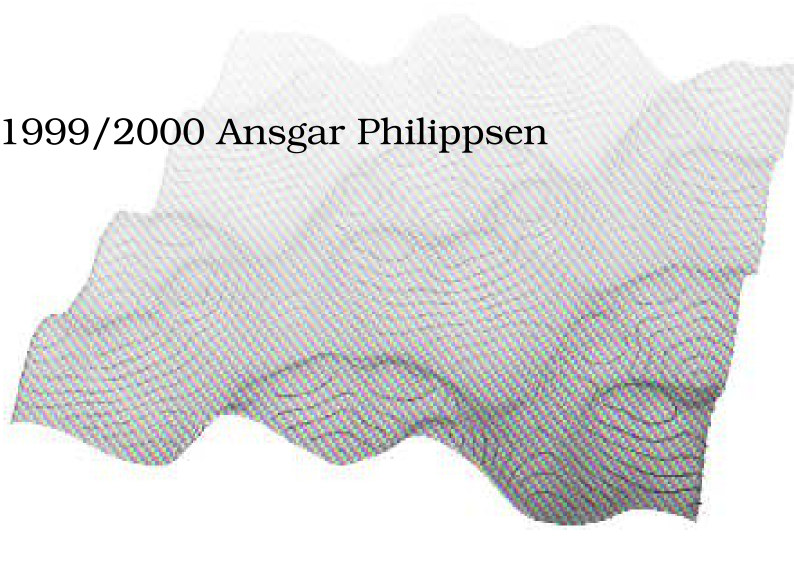


Table of Contents

List of Tables

Thank you for your interest in DINO. This program aims to participate in the ongoing effort to understand the structure and function of biological macromolecules. It seeks specifically to visually combine results obtained from all different fields that involve structural biology, such as x-ray crystallography, structural NMR, electron and atomic-force microsopy and bioinformatics (including molecular dynamics, structure prediction and electrostatic potential calculations).

Please note that the current release is considered beta[1]. Most of the features have been implemented, however, and current work is dedicated to improving code robustness, removing bugs and implementing more formats. Please do not hesitate to contact me (`dino@bioz.unibas.ch`) for any questions, comments, criticism or bug reports.

The DINO homepage is at `http://www.bioz.unibas.ch/~xray/dino`

Acknowledgements:

Raimund Dutzler, Joerg Stetefeld, Paola Storici, Guido Capitani, Tilman Schirmer, Dimitrios Fotiadis, Daniel Mueller, Andreas Engel, Gitay Kryger, Harry M Greenblatt, Kaspar Locher, Ralf W. Grosse-Kunstleve, Paul Adams, Michael L. Connolly, Don Bashford, Mathew Caughron, Todd M Serulneck, Eugenio Santenelli, Domenico Bordo

1. beta is only a state of mind

1 Quickstart

This chapter will quickly go through a typical DINO script, introducing concepts and leaving details to the following sections of the manual. The files used here are the sea-hare myoglobin structure 1MBA (Bolognesi et. al., 1990) and the corresponding MSMS surface files (to be found on the DINO homepage).

Start up DINO in the directory with the files and enter the following commands at the prompt `dino>`

```
load myo.pdb
```

The structure in the file `myo.pdb` is loaded from disk. This creates a dataset called `myo` which is used in subsequent commands to adress the structure.

Nothing is visible on the screen yet. We first need to create an object which will be displayed in the main graphics window. To do this, the dataset command `new` is used. We will name the object `all` because we do not apply a selection, hence the object contains all atoms of the structure:

```
.myo new -name all
```

There is *still* nothing visible: we need to center the object on the screen. This is accomplished with the scene command `center`, using a recursive subprompt. Right now, it is important to realize that `[.myo]` returns the geometric center of the structure:

```
scene center [.myo]
```

The front and back clipping planes are adjusted:

```
scene autoslab
```

The protein should now be visible and react to input from the mouse; hold the left mouse button down in graphics window and move the mouse or try the dials (if present).

Let us create a specific object, e.g. just the heme group. This time `new` is called with a selection:

```
.myo new -name hem -sel rname=HEM
```

The graphical appearance is modified with the object command `render`:

```
.myo.hem render custom,sr=0.35,bw=0.15
```

We can center on the object or on a specific atom, in this case the heme iron

```
scene center [.myo.hem]      // center on object
scene center [.myo:148.FE] // center on atom
```

Try these different render modes and parameters

```
.myo.hem render cpk
.myo.hem render detail=2
.myo.hem render detail=6
.myo.hem render simple
```

Since we have the heme as an individual object, we can modify our original object `all` to *not* have the heme anymore by using the object command `renew` combined with a selection:

```
.myo.all renew -sel not rname=HEM
```

To change colors of individual atoms, the object command `set` with a selection is used, e.g. to color all carbons in green:

```
.myo.all set color=green -sel aname=C*
```

We will look at more syntax. First we turn off the display of the protein object

```
.myo.all hide
```

Now we will generate a sphere of residues surrounding the heme. We create a new object and use the within operator <> together with an object as selection criteria. Additionally - since we already have the heme as an object - we exclude the heme from the selection

```
.myo new -name sphere -sel 10<>.myo.hem and not rname=HEM
```

The selection mechanisms has picked out some individual atoms which are not connected to anything and are displayed as crosses. To tell DINO that we want to select residues, change the selection mode of the dataset to `residue` (default is `atom`), and simply renew the object, it will remember its selection syntax:

```
.myo set smode=residue
.myo.sphere renew
```

Let us turn to the other object type for structure datasets: the trace. We will first hide the sphere object

```
.myo.sphere hide
```

A trace object is created and colored in purple:

```
.myo new -name ca -type trace
.myo.ca set color=purple
```

trace objects can also be rendered in different ways, e.g. as interpolated tubes:

```
.myo.ca render tube,bw=0.3
scene center [.myo.ca]
```

try also following render modes and parameters

```
.myo.ca render sline
.myo.ca render tube,bw=1.0
.myo.ca render simple
```

This concludes the structure dataset part, we will go on to the surface:

```
.*.* hide  // hide everything
```

A surface generated with MSMS is loaded and named `surf`. Then a surface object is created, centered on the screen and the clipping planes are adjusted

```
load myo -type msms -name surf
.surf new -name all
scene center [.surf.all]
scene autoslab
```

To do more with the surface, we can attach it to a structure dataset:

```
.surf attach .myo
```

Now each surface vertex (point) carries information about the atom closest to it, so we can use atom-selection statements for the surface object

```
.surf.all set color=blue -sel rname=LYS,ARG
.surf.all set color=red -sel rname=ASP,GLU
```

A second surface was calculated, without the heme group present in the structure. This surface is loaded, a new object is generated and centered on the screen:

```
.*.* hide
load myo_apo -type msms -name surfa
.surfa new -name all
scene center [.surfa.all]
```

Since the surface was calculated without the heme, we will turn the heme object back on and try to find it in the structure by rotating the surface around.

```
.myo.hem show
.myo.hem render custom,sr=0.3,bw=0.3,detail=3
```

To take a closer look on the surface that surrounds the heme, we will create a new object: the part of the surface that lies within 10 of any atom of the heme

```
.surfa.all hide
.surfa new -name hem -sel 10<>.myo.hem
```

We can now turn on the sphere object and render the surface transparent for a nice view:

```
.myo.sphere show
.surfa.hem render t=0.5
```

The new surface can also be attached to the structure

```
.*.* hide
.surfa.all show
.surfa attach .myo
```

To see what has happened during the attachment, we will color the complete surface in yellow that is part of the protein:

```
.surfa.all set color=yellow -sel $protein
```

In the groove of the heme the surface is partially white. The reason for this is that the heme was included in the attachment. To avoid this the structure dataset is restricted, and then the attachment is performed, ignoring all restricted atoms. The surface is un-attached first.

```
.myo restrict not rname=HEM
.surfa attach none
.surfa attach .myo
.myo restrict
```

Same command as above, but now the complete surface should be yellow

```
.surfa.all set color=yellow -sel $protein
```

This concludes the quickstart. Please visit the DINO homepage for more tutorials.

2 Overview

2.1 Structural Data

The different research fields generate different types of structural data. In x-ray crystallography, the experimental electron density is interpreted to yield a model with atomic coordinates and chemical connectivity. Structural NMR gives rise to an ensemble of structures, also in the form of atomic coordinates. Electron microscopy generates 2D images and reconstructed 3D density, while atomic force microscopy (AFM) offers surface topography information. Bioinformatics produces structural data as well: molecular dynamics and related methods in biophysical chemistry generate trajectories of atomic models or electrostatic potentials, other calculations result in the molecular or solvent accessible surface of a macromolecule.

DINO aims to combine the diverse structural data by implementing four different data types:

- **Structure**: Coordinates of atoms with chemical connectivity, optionally organized in residues, which in turn are optionally organized in chains. For ensembles of structures, a model number is present as an additional hierarchy. For trajectory data, each frame can be addressed with a number.
- **Scalar Field**: 3-dimensional array of scalar values, representing e.g. electron density or electrostatic potential.
- **Surface**: Collection of faces (triangles) build from list of vertices (points in 3D space), approximating the surface.
- **Surface Topography**: 2-dimensional grid, each grid point carries a height value.

2.2 Startup

DINO supports a number of command line arguments:

```
dino [-s scriptfile] [-log logfile] [+log] [-nostereo] [-debug]
```

`-s scriptfile`: The file `scriptfile` is called immediately after startup, just as if `@scriptfile` had been entered as the first command.

`-log logfile, +log`: Per default, a file called `logfile.dino` is written out continuously with all issued commands. The name of the logfile can be changed with `-log`, or logging can be disabled completely with `+log`.

`-nostereo`: In certain situations in can be necessary to disable stereo detection.

`-debug`: will print out lots of debugging messages.

Upon startup, DINO will try to autodetect the stereo hardware and any additional input devices and print this along with the version info and the graphics hardware ID. A typical output might look like this:

```
Welcome to dino v0.8.0   (http://www.bioz.unibas.ch/~xray/dino)

HighEnd stereo detected
Dialbox detected
Spaceball detected
Graphics Subsystem ID: SGI RES/S/1/2
OpenGL Version 1.1

dino>
```

With the prompt awaiting input.

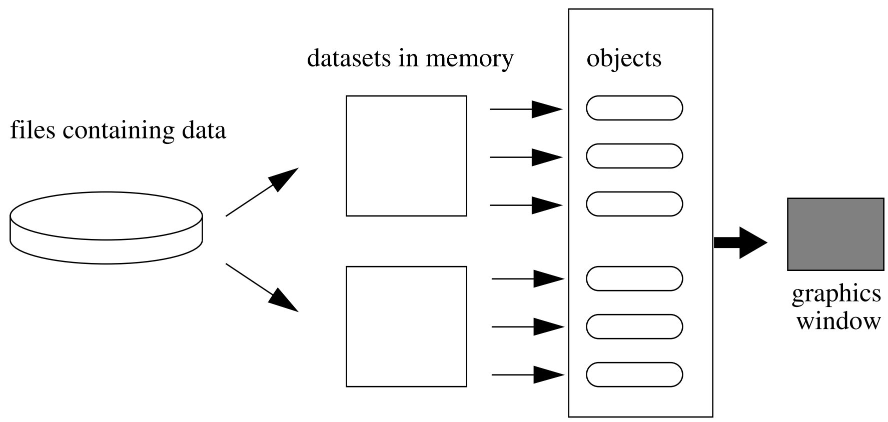
2.3 Concepts

2.3.1 Interaction

There are two levels of interaction with the program. One is via the graphical user interface, including the main graphics window and the object menu. The other is through the prompt, where commands can be typed in. The shell handles and interprets all typed input.

2.3.2 Dataset and Object

The data is loaded from a file into the program and is stored in memory. This memory block is called a *dataset*. From the dataset one or more *objects* are created, these objects appear on the graphics screen and can be interactively manipulated. A selection mechanism allows to specify only a subset of the data; this can be used during object creation or for modifying the objects (e.g. recoloring). A particularly powerful feature of the selection is called cross-selection, in which the selection is based on objects or properties from other datasets.



2.3.3 Special syntaxes

color

A color can be defined in two ways:

- a color name as defined by the X-windows system. The UNIX command `showrgb` will list all available color names. A blank in the color name must be replaced with an underscore, e.g. `light steel blue` becomes `light_steel_blue`. Example: `black white red blue green yellow cyan magenta medium_spring_green`
- an explicit rgb triplet in the form `{r,g,b}`, where `r` denotes the red component from 0 to 1, `g` the green component from 0 to 1 and `b` the blue component from 0 to 1. Example: `{0.5,0.7,0.9} {1,1,0}`

vectors and matrices

When dealing with position or transformation, it is necessary to specify vectors or matrices. One-dimensional vectors and two-dimensional matrices are specified as follows.

$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ is written as `{x,y,z}`, a 3x3 matrix $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$ is written as `{{a,b,c},{d,e,f},{g,h,i}}`

and a 4x4 matrix accordingly.

2.3.4 Manual Naming and Conventions

This manual is evolving along with the code. I realize that some parts miss deeper explanation and/or examples. Any comments and criticisms regarding the manual are just as welcome as comments and criticisms regarding DINO itself. More examples are provided on the dino homepage.

Syntax and example

Throughout the manual, syntax and examples are given in the form

Syntax: `this is a syntax statement`

and

Example: `this is an example of an example`

The use of square brackets `[]` in the syntax describes optional parts, while square brackets in the example are meant literally.

PV

Throughout the manual, the expresion `PV` denotes a Property-Value pair, in the form:

Syntax: `Property Op Value,[Value2 . .]`

This is used during value assignement, comparison and parameter setting. During comparison, mutliple values can be given as a comma separated list. For boolean properties (meaning they are either True or False), `Op` and `Value` can be ommited, setting the property to True or - when prefixing it with an exlamation mark - to False.

Example: `bg=white     rnum=10,14,20,33     v>2.0     !depthc`

3 Shell

The DINO shell provides an interface between program and user, just as the UNIX shell provides an interface between operating system and user. The DINO shell offers a prompt in the terminal, where commands can be typed in. In essence, the input is parsed, some syntax rules are applied and the command line is broken down into words. The first word is checked against the internal shell commands, if no match is found the command line is passed to the database manager.

3.1 Syntax

Variables

Words beginning with a `$` are interpreted as variables and are replaced with the value of the variable. (see commands `set` and `unset` below).

Recursive Subprompts

An expression enclosed in square brackets is considered a subprompt. It is evaluated first, and the result (return value) of the sub-expression is inserted at the position of the subprompt. Consider the following input:

```
> expression1 [expression2 arg]
```

`expression2 arg` is treated as a separate command, its result `result2` is inserted in the place of the original subprompt and becomes:

```
> expression1 result2
```

The subprompts can be nested (almost) unlimited.

Scripts

A script is called with the `@` sign, followed by the filename. Additional words are interpreted as parameters and are passed as variables to the script, accessible as `$1`, `$2` ... `$n` (n = number of arguments).

Syntax: `@scriptfile [param1 ...]`

Comments

Two slashes denote a comment (C++ style) and can appear anywhere in the line, with the rest of the line (up to newline) treated as a comment.

Example: `// this is a comment`

Character Protection

Use the backslash to protect the following character from being interpreted by the shell parser. This is useful when continuing a line in a script or literally inserting a $ or square bracket.

Example:
```
// a line can be continued if the \
last character before newline is '\'
```

Splitting Input

Multiple command lines can be entered on a single line when separated with a semicolon:

Syntax: `command1 arg1 ; command2 arg2 ; command3 arg3`

3.2 Shell Commands

Writing Output to the Terminal or a File

The command `echo` prints all of its arguments to the terminal. The output can be redirected into a file with > (write) or >> (append):

```
Syntax: echo WORD1 [WORD2 ...] [>[>] file.out]
```

Variables and Aliases

Variables are defined and removed with `set` and `unset`. The command `var` will list all currently defined variables with their values.

```
Syntax: set VAR VALUE
        unset VAR [VAR ...]
```

An alias is a special form of variable. It is defined with `alias` and removed with `unalias`.

```
Syntax: alias ALIAS EXPRESSION
        unalias ALIAS
```

If the first word on the command line (or the first word in a subprompt!) matches `ALIAS`, it will be replaced by `EXPRESSION` (which can consist of multiple words). `alias` by itself will list all defined aliases.

Pausing Script Execution

In a script, the command `pause` will stop execution until a key is pressed. If the key is ESC, script execution is stopped.

Stopping Script Execution

In a script, the command `break` will discontinue script execution and return one level up, either to the calling script or the command prompt.

Executing a Unix Command

Use `system` or `!` to execute a command in a unix shell

```
Example: system gimp pic1.png
         !ls -l
```

Exiting

To quit DINO use `exit`. To exit DINO use `quit`[1].

3.3 RPN Stack

The commands `push`, `pop`, `peek`, `opr`, `clear`, `dup` and `show` manipulate the arithmetic RPN stack[2]. `push` moves all arguments onto the stack (from left to right). `pop` retrieves values from the stack, optionally into supplied variable names. `peek` returns the topmost value without removing it from the stack. `opr` takes a list of operator which are applied (left to right) to the stack. `clear` empties the stack. `dup` duplicates the top value. `show` displays the complete stack.

---

1. or `stop bye adios ciao finish terminate`
2. In a nutshell: In RPN (Reverse Polish Notation), a stack is filled with values, and an operator is called that uses the topmost value (unary op, e.g. sqrt) or the two topmost values (binary op, e.g. +), placing the value back on the stack. A simple arithmetic expression as 1+1 becomes 1 1 +, a more complex as (1+2)*(4/5+6) becomes 1 2 + 4 5 / 6 + *. People used to the HP calculators or programming PS will be familiar with this approach.

```
Syntax: push VAL [VAL ...]
        pop [VAR1 [VAR2 ...]]
        peek
        opr OPR
        clear
        dup
        show
```

The availabe unary and binary operators are shown in Table 1 (P. 11).

Table 1: Unary and Binary Operators of Shell RPN Stack

unary operator	effect
`+-`	changes the sign on scalars, elements of vector or matrix
`inc dec`	increases, resp. decreases scalar, elements of vector or matrix by one
`abs`	return absolute of scalar or length of vector, invalid for matrices
`inv`	inverse of scalar, invalid for vectors and matrices
`log ln exp sqrt`	calculates the decimal resp natural logarithm, exponential function or square root of scalar, invalid for vector or matrix
`sin cos tan asin acos atan`	trigonometric functions for scalars, invalid for vector or matrix
`int float`	returns the integer part or float, or forces return of float
`det`	returns determinant of matrix, invalid for scalar and vector
binary operator	**effect**
`+ -`	basic addition and subtraction, vectors and matrices must have identical elements
`*`	multiplication, valid combinations: scalar*scalar, scalar*vector, scalar*matrix, vector*vector (dot product), vector*matrix, matrix*matrix
`/`	simple division, valid combinations: scalar/scalar, vector/scalar, matrix/scalar
`pow`	power of x (1st position) to y (2nd position)
`x`	calculates cross product between two vectors
`rmat`	Requires a direction vector V and a scalar value S, returns the rotation matrix of a S degree rotation around axis V

3.4 Pre-defined variables and aliases

aliases

`stereo: scene stereo`

`mono: scene mono`

`write: scene write`

variables

```
protein: (rname=ALA,CYS,ASP,GLU,PHE,GLY,HIS,ILE,LYS,LEU,
               MET,ASN,PRO,GLN,ARG,SER,THR,VAL,TRP,TYR)
```

```
dna: (rname=A,C,G,T)
```

```
rna: (rname=A,C,G,U)
```

```
aliphatic: (rname=ALA,GLY,ILE,LEU,MET,PRO,VAL)
```

```
aromatic: (rname=PHE,TYR,TRP)
```

```
hydrophobic: (rname=ALA,VAL,PHE,PRO,MET,ILE,LEU,TRP)
```

```
basic: (rname=ARG,LYS)
```

```
basic2: ((rname=LYS & aname=NZ) | (rname=ARG & aname=NH1,NH2))
```

```
acidic: (rname=ASP,GLU)
```

```
acidic2: ((rname=GLU & aname=OE1,OE2) | (rname=ASP & aname=OD1,OD2))
```

```
polar: (rname=SER,THR,TYR,HIS,CYS,ASN,GLN)
```

```
polar2: ((rname=SER & aname=OG) | (rname=THR & aname=OG1) |
         (rname=TYR & aname=OH) | (rname=HIS & aname=ND1,NE2) |
         (rname=CYS & aname=SG) | (rname=ASN & aname=OD1,ND1) |
         (rname=GLN & aname=OE1,NE1) | (rname=TRP & aname=NE1))
```

4 Scene and GUI

The expression scene refers to the contents of the main graphics window. In this section, interaction with the graphics window using the mouse and other input devices will be explained as well as the scene commands which allow customization of the global graphical properties. In addition, the (slim) graphical user interface will be discussed.

4.1 Scene Interaction

The viewpoint (camera position) can be interactively changed. DINO supports a number of input devices. Each input device can be characterized as having one or more axis, zero or more buttons and a combination of keyboard modifiers (SHIFT, CTRL, ALT) pressed.[1] The following input devices are supported:

mouse

The mouse has two axis (left-right and up-down) and is expected to have three buttons (left, middle, right). The right button is reserved for the context-sensitive submenu (see “4.12 The GUI” on page 20).

Clicking[2] on an atom with the left mouse-button will 'select' this atom:

- The name of the atom is displayed in the status bar (bottom of graphics window)
- Its name is stored in the shell variable CS (for Current Selection) and at the same time is pushed onto the scene stack (see below).

Additionally, if SHIFT is pressed while clicking the atom, the atom label will toggle (first click show, second click hide ...). Depressing the left mouse-button longer than necessary for a click will no longer count as a click but as pressed. Table 2 (P. 13) shows the default mouse settings.

Table 2: Mouse Mappings in Scene window

combination	function
`LeftButton`	rotate around x and y
`MiddleButton`	translate along z (slow and fast)
`LeftButton + MiddleButton`	rotate around z
`LeftButton + SHIFT`	translate along x and y
`MiddleButton + SHIFT`	move slab along z
`LeftButton + MiddleButton + SHIFT`	change slab width

spaceball

A spaceball is a highly recommended, but slightly expensive input device that incorporates all six rotational and translational axis in one device. For exploring data, it is much more convenient than either the dialbox or the mouse. There is a spaceball that works on linux[3].

---

1. In the current version the modification mapping of input is not supported, but this should change in a future version, so that the input is fully customizable.
2. Clicking in this context means pressing and releasing the button within 200 milliseconds

Table 3: Dialbox Mappings in Scene window

slab width	slab translate
rotate z	translate z
rotate y	translate y
rotate x	translate x

dialbox
A standard dialbox has 8 dials (equals 8 axis). Rotation around and translation along x,y and z are mapped to six of these dials. The remaining two manipulate the front and back clipping planes (slab). One translates the slab along z, the other changes its width. For the standard SGI dialbox the mappings are displayed in Table 3 (P. 14) (see also "`man dialwarp`").

4.2 Writing a snapshot of the current scene

The scene command write will save the current scene to a file in a variety of formats.

Syntax: `[scene] write file.ext [-scale SCALE] [-accum N]`

Because of its frequent usage, an alias has been automatically created, so `scene` can be omitted. The format is guessed from the extensions:

Table 4: scene write output formats

extension	description
.png	standard PNG format, picture is an exact screen copy, usually smaller than TIFF file.
.tiff	standard TIFF format, picture is an exact screen copy
.pov	POVray (`www.povray.org`) output
.r3d	Raster3D format[a]
.ps	PostScript format
.rgb	SGIs RGB format, **no longer supported in v0.8!**

a. Merrit & Bacon (1997) Meth. Enzymol. 277 `http://www.bmsc.washington.edu/raster3d`

`.tiff` & `.png`: The `-scale` and `-accum` parameters are only applicable to TIFF and PNG output `-scale` is either an absolut number or an expression in percent,designated by a percentage sign. `-accum` will use the accumulation buffer to generate `N` images that are slightly offset from each other, causing an overall smoothing[1]. Supported values for N are 1(off), 2, 3, 4, 5, 6, 8, 9, 12, 16. The images are generated with offline rendering through an X-Windows pixmap. Since the largest

3. The spaceball is called Magellan(TM), is sold by Logitech(TM), and such a device was kindly lend to the author for implementing the spaceball code into the linux version of DINO by logicad3d(TM) (`http://www.logicad3d.com`).
1. From OpenGL Programming Guide 3rd Ed. *Scene Antialiasing* p. 451

pixmap size is X-Windows implementation dependent and also depends on the amount of RAM, a huge output file might crash the X-Server. 2000x2000 should work in most cases.

`.pov`: *Persistance Of Vision* is a powerful, generic raytracer. DINO writes two output files, one with extension `.pov` and one with extension `.inc`. The `.pov` file is meant to be user-editeable and it contains the lighting, fog and viewpoint definition as well as variables to change object properties, i.e. the applied texture, transparency etc. Stereo pairs can be generated with the same POV output, the `.pov` file contains information about this. For more information, please refer to the POV tutorial on the DINO homepage.

`.r3d`: *Raster3D* is another raytracing package, aimed specifically at protein structure. Note that Raster3D output does not preserve the perspective, so stereo pairs cannot be generated this way (see also scene set view below).

`.ps`: PostScript Level 2 output, transparency is not supported, and smoothly shaded triangles are approximated by subdivision into smaller, unicolored triangles. This format is suitable for schematic line drawings.

Example:
```
write pict1.png -s 1000 -a 4
write scene.pov
```

4.3 Setting scene properties

The scene command `set` allows the modification of scene properties, while `get` retrieves them:

Syntax:
```
scene set PV[,PV...]
scene get P
```

See Table 5 (P. 16) for a list of all scene properties.

Examples

white background
```
scene set bg=white
```

generating stereo figures
```
scene set view=left // left view
// increase -s number to get higher resolution
write l.png -s 1000
scene set view=right // right view
write r.png -s 1000
scene set view=center
// use montage or something to paste the two files l.png and r.png
// together, print them at a width of 13cm and you are done. I use
// the ImageMagick command:
// montage +frame +label +shadow -geometry 100%
//         -tile 2x1 l.png r.png s.png
// for cross-eyed stereo, swap left and right view
```

writing the current orientation matrix
```
echo [scene get mmat] > view
// into file 'view' (e.g. to paste into a script)
```

writing the current orientation - more 'sophisticated'
```
echo scene set mmat=[scene get mmat] > view
echo scene set near=[scene get near] >> view
echo scene set far=[scene get far] >> view
// after some commands, in new script, etc...
@view
// restores orientation and clipping planes from file view!
```

Table 5: Scene Properties

property	explanation	default
bg	Background color. Accepts a colorname or an explicit {r,g,b} value	black
center	Use with get to retrieve the current center of rotation. Set the center of rotation with the shortcut scene center	{0,0,0}
depthc	If this flag is true, depthcueing between the near and far clipping planes - modified with fogd - will be performed.	true
dither	boolean flag, if set will enable global dithering on displays with less than 24 bits of color buffer	true
eyedist	This float denotes the occular distance for stereo projection	150.0
far	z value of far clipping plane. This float should always be larger than 0 and always larger than near. Op can be =, += or -=	1000.0
fixz	If this flag is true, the clipping planes will move along any z-translation	true
fogd	Offset from fog to far clipping plane. Values are relative to (far-near) and can range from -1.0 to +1.0	0.0
fov	Field-of-view angle for perspective projection	25 (degrees)
mmat	Current modelview matrix (rotation and translation in one 4x4 matrix). See examples below	
near	z value of near clipping plane. This float should always be larger than 0 and always smaller than far. You can use the operators += and -= to increase/decrease the value instead of setting it explicitly with =	1.0
ortho	Use this flag to activate an orthographic viewing matrix. The aspect ratio is automatically determined from the window size	no
persp	Use this flag to activate a perspective viewing matrix with a field-of-view angle of fov. The aspect ratio is automatically determined from the window size	yes
rot	Current 3x3 rotation matrix	identity
slabw	Width of slab (distance netween near and far clipping plane)	999.0
trans	Current translation vector as {x,y,z}	{0,0, -100}
view	This property can take any of the three values left, right, center (default). view=left resp. view=right switch to the left resp. right stereo viewpoint. This allows easy generation of stereo pictures (see example)	center

4.4 Stereo Mode

On graphics hardware that supports stereo viewing[1] the scene command `stereo` will toggle between stereo and mono mode.

Syntax: `[scene] stereo`

Due to its frequent usage, an alias is automatically created so `scene` can be omitted. To force stereo off, use `stereo off` or `mono,` to force it on use `stereo on`

1. Currently only SGI hardware stereo is supported in DINO

```
Syntax: [scene] stereo off
        [scene] mono
        [scene] stereo on
```

Stereo hardware is a confusing and complicated chapter in its own. Please refer to the DINO homepage for machine specific problems and solutions.

For cross-eyed stereo display - also called split-screen stereo - use the `split` command to toggle between split-screen and normal stereo. Note that while in split-screen mode, `scene write` will write out the left and right image into a single file.

```
Syntax: scene split
```

4.5 Manipulating the scene

As mentioned above, rotation, translation and slabbing can be controlled by input devices. There are also some commands for this purpose:

```
Syntax: scene center {x,y,z}
        scene autoslab
        scene transx VALUE
        scene transy VALUE
        scene transz VALUE
        scene transm {x,y,z}
        scene rotx VALUE
        scene roty VALUE
        scene rotz VALUE
        scene rotm {{a,b,c},{d,e,f},{g,h,i}}
        scene reset
```

`center`: The center of rotation is set with center. This is usually combined with a recursive subprompt in the form:

```
Example: scene center [.ds] // center on dataset
         scene center [.ds.obj] // center on the object
```

`autoslab`: moves the near resp. far clipping plane exactly to the boundaries of the currently visible objects.

`trans* rot*`: translates along resp. rotates around the given axis, while the matrix version (`transm` / `rotm`) applies a matrix to the current translation / rotation matrix[1].

`reset`: returns the scene to its default state[2].

The command `grab` will regain input from an input device (see “4.10 Input” on page 20).

```
Syntax: scene grab INPUTDEVICE
```

4.6 Current Point

Everytime the left mouse button is clicked in the graphics window, a single coordinate is stored in the shell variable `$CP`. The position of the mouse alone can only determine two out of three coordinates components: one can envision an infinite line perpendicular to the screen going through the mouse pointer. `$CP` is determined by calculating the intersection of this line with the near and far clipping

1. The rotation matrix is not verified (-> determinant==1) ! Expect weird things when using arbitrary values
2. transmat={0,0,-100}, rotmat=identity ({{1,0,0},{0,1,0},{0,0,1}}), near=1.0, far=1000.0

plane and then using the middle of these two point. This is useful when looking at scalar fields, e.g. electron density, with a small slab width: Click anywhere into the density and call
`scene center [$CP]`.

4.7 Scene Stack

The program maintains a history of the selection. Every time an atom is selected in the graphics window, its name is pushed onto the scene stack. Following are commands to manipulate this scene stack.

```
Syntax: scene push ARG
        scene pop
        scene peek
        scene clear
```

`push`: pushes ARG onto the stack
`pop`: retrieves the topmost value and removes it
`peek`: returns the topmost value without removing it
`clear`: erases the stack.
Why would this be useful ? As an example: Distance measurement of two consecutively selected atoms:

```
Example: push [[scene pop]]
         push [[scene pop]]
         opr - abs
         scene message Distance: [peek] Angstrom
```

The first two lines retrieve the selected atoms from the scene stack with `[scene pop]` and insert the name of the atom in the line. Because this result is again embraced with `[]`, it defaults to `get xyz` (see below). These coordinates are pushed onto the arithmetic shell stack, the difference is computed (`opr -`)and then the length of the resulting vector (`opr abs`). The result is `peek`ed of the shell stack and displayed in the status bar of the graphics window. This is exactly how the command `dist` in the user menu is implemented.
Another example: centering on the middle of two atoms consecutively selected atoms

```
Example: push 0.5 [[scene pop]] [[scene pop]]
         opr + *
         scene center [pop]
```

4.8 Lighting

DINO supports the use of up to 8 lightsources. They are addressed as

```
Syntax: scene:lightn command [parameters]
```

where *n* is a number from zero to seven. Following is a list of commands to modify the lightsource settings, refer also to Table 6 (P. 19) for light properties.

- `set get`: modify or retrieve light properties
- `on off`: shortcut for `set on` resp. `set off`
- `local`: shortcut for `set local`
- `global`: shortcut for `set global`
- `show`: display information about light

The lighting contribution to each vertex is calculated from the direction of the lightrays emitted by a lightsource. The lightrays of a directional lightsource are all parallel. With a positional lightsource, however, the direction of the lightrays is different for each relative position of vertex and lightsource. In real world terms, the sun can be considered a directional lightsource, while a lamp is a positional one.

A global light is unaffected by the scene transformation, while a local light is rotated and translated along with the scene objects.

Attenuation is only valid for positional lights. It mimics the dimming of the light in respect to distance from the lightsource using the following formula, where f is the factor applied to the light-contribution, d is the distance of the vertex to the lightsource and kc, kl and kq are the attenuation properties:

$$f = \frac{1}{k_c + k_l d + k_q d^2}$$

Example:
```
// place a local, positional light at {0, 0, 10}
// with a linear attenuation of 0.1
scene:light1 set pos={0,0,10,1}, local, kl=0.1
scene:light1 on
```

For more information about lighting, please refer to an OpenGL manual (see reference section)

Table 6: Light Properties

property	explanation	default light 0	default light 1-7
`on off`	flag if light is on of off	on	off
`local`	light position will move with the camera		
`global`	light position is fixed	X	X
`pos`	vector is form {x,y,z,w}, where x,y and z are coordinates in space and w=0 denotes directional light and w=1 positional light	`{0.1,0.2,1.0,0.0}`	`{0.0,0.0,1.0,0.0}`
`amb`	ambient color contribution, can be given as scalar grayscale value or as color `{r,g,b}`	0.05	0.0
`diff`	diffuse color contribution, can be given as scalar grayscale value or as color `{r,g,b}`	0.6	1.0
`spec`	specular color contribution, can be given as scalar grayscale value or as color `{r,g,b}`	0.3	1.0
`kc`	constant attenuation factor	1.0	1.0
`kl`	linear attenuation factor	0.0	0.0
`kq`	quadratic attenuation factor	0.0	0.0
`spotc`	spot cutoff angle (0 to 90), 180 means no spot	180.0	180.0
`spote`	spot exponent	0.0	0.0
`spotd`	spot direction	`{0,0,-1}`	`{0,0,-1}`

4.9 Arbitrary Clipping Planes

In addition to the front and back clipping planes, an arbitrary clipping plane can be turned on. A plane is defined by an origin (`pos`) and the direction (`dir`) of the plane normal

```
Syntax: scene:clipN [on|off],[pos={x,y,z}],[dir={xn,yn,zn]
        scene:clipN grab DEVICE
```

`N` is the number of the clipping plane, currently only `clip0` is implemented. The position defaults to `{0,0,0}`, the direction defaults to `{0,0,1}` A clipping plane can grab an input device to be rotated and translated arbitrarily (see “4.10 Input” on page 20).

4.10 Input

As mentioned above, there are three input devices supported in DINO: mouse, dials and spaceball. Internally, there are two input 'tables' for each device, called `mouse` and `mouse2`, `dials` and `dials2` and `spaceball` and `spaceball2`. The `*2` version is activated when `cntrl` is held down on the keyboard. Why would this be useful ? Each dataset as well as the scene contain a `grab` command, which reroutes the input info from an input device to that dataset or the scene. After startup, all input is routed to the scene, causing the camera position to be modified. If an input is routed to a dataset, however, its rotation/translation is modified, effectively leading to a rigid body rotation within the global coordinate system. In a situation where dataset A has grabbed `mouse`, the scene would still get input from `mouse2`, so pressing `cntrl` while moving the mouse would cause a camera update and not a transformation of dataset A. This also applies to the dials and the spaceball.

4.11 Odds and Ends

A string can be displayed in the status bar of the graphics window:

```
Syntax: scene message MESSAGE
```

The command `spin` will toggle spinning of the objects, i.e. the continuation of the last mouse-induced rotation. False per default.

```
Syntax: scene spin
```

`bench` will continuously force screen updates and allows simple 'benchmarking' by displaying the fps in the status bar. Call `bench` again to turn off.

```
Syntax: bench
```

4.12 The GUI

The GUI in DINO is implemented only very rudimentary, a situation which will be improved upon as the program matures.

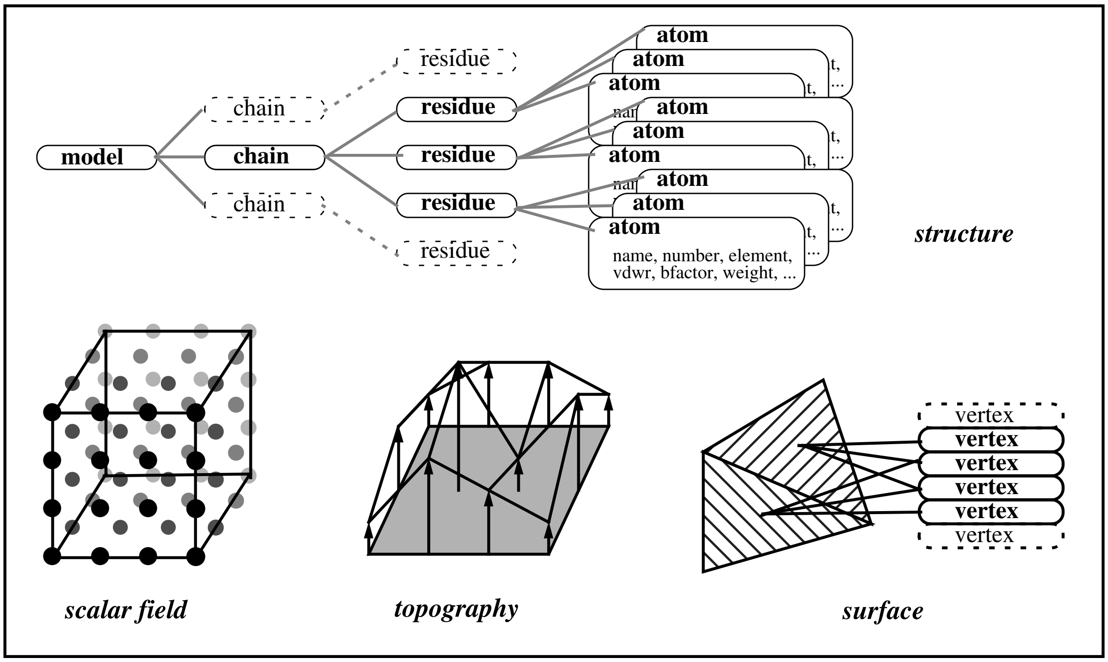
What is present so far is an object menu, which appears as a second window after startup. It contains a button *scene*, and will fill itself when datasets are loaded and objects created. Left-clicking on an object name will toggle its display, while right-clicking on an object-name, dataset-name or scene will cause a (context sensitive) submenu to appear.

In addition, the functionality of a user menu is implemented, but it is currently not customizable. It appears (as a popup) when right-clicking somewhere in the graphics-window.

5 The Datasets

5.1 Common Concepts

The structural biology data consists of basic elements which are specifically related to each other. In the structure dataset, the basic element is the atom, which is usually organized into residues, which in turn are optionally organized as chains, which in turn are optionally organized into models. In the scalar field and the surface topography, each grid point is the basic element, arranged into a regular 3-dimensional array (scalar field) or 2-dimensional plane (surface topography). The basic element of the surface is the vertex (coordinate in 3D space), which is organized into faces (triangles). A key property of all elements - apart from dataset specific ones - is that they all have a defined position in space.



A dataset can be thought of as a block of memory that holds the data. To visualize the data, objects are created from the dataset, which are displayed in the graphics window. Just as there is no limit to the number of datasets that can be loaded, there is no limit to the number if objects that can be created from a single dataset.

Now we have three different layers of the data: the dataset, the object and the individual data element. The object and the element always belong to a specific dataset. The three layers are addressed (given commands) in the following way:

dataset: the name of the dataset, preceded with a dot, then the command with its (optional) parameters

```
Syntax: .dataset command [parameters ...]
```

object: dot-datasetname-dot-objectname command (optional) parameters

```
Syntax: .dataset.object command [parameters ...]
```

data element: dot-datasetname-colon-element command (optional) parameters. This is only implemented so far for the structure dataset (see below)

```
Syntax: .dataset:element command [parameters ...]
```

Wildcards (* and ?) can be used in the dataset and object names to match multiple datasets resp. objects. For example, to hide all objects with one command use

```
Example: .*.* hide
```

5.1.1 Prototype

The following scheme outlines the basic steps for displaying your data:

- load a file, its content is stored as a dataset
- create a new object of a specific type from the dataset, optionally applying a selection and/or setting object properties
- center object on the screen
- modify object properties with set, optionally applying a selection and/or a range criteria
- change the rendering state

```
Example: load myo.pdb
         .myo new -name prot -sel rnum=1:146
         scene center [.myo.prot]
         .myo.prot set color=cyan -sel aname=C*
         .myo.prot render custom,sr=0.25,bw=0.15
```

5.1.2 Creating Objects

After a dataset is loaded objects can be created from it. This is accomplished with the dataset command `new`. During object creation the name and type of the object as well as object properties can be set and a selection can be applied.

```
Syntax: .dataset new [-name N] [-type T] [-selection SEL] [-set S]
```

As a default, the name of the dataset is used and all elements are selected. The default type and object properties depend on the dataset (see below).

5.1.3 Properties

The scene, dataset, objects and elements are all characterized by properties, e.g. the background color, the selection mode, the object color, the position in space, etc.

When an object is created from a dataset, some element properties are shared and some are copied. To retrieve the shared properties, the data-element command `get` is used. To modify the shared properties, the dataset command `set` is employed, optionally with a selection. To modify a copied property, the object command `set` or the dataset command `set` is used; usage on the dataset level will change the default.

```
Example: // retrieve shared property (e.g. bfac)
         .struct:A.114.CA get bfac

Example: // set shared property (e.g. rtype)
         .struct set rtype=helix -sel rnum=1:14,20:35

Example: // set copied property (e.g. color)
         .struct.obj set color=green -sel ele=C  // for object only
         .struct set color=green -sel ele=C   // default coloring
```

The different properties are described in tables below, specifiying wether they belong to the dataset, object or element.

5.1.4 Selection Syntax

DINO features a powerful selection syntax. A selection consists of one or more PVs connected by logical operators `and` (or `&`) and `or` (or `|`) , optionally grouped by parenthesis ( ). A PV is in principle build up from a property, and operator and one or more (comma-separated) values and can be lead by a `not` (or `!`).

There are three principle types of PV used for selection:

- *standard*: property is one of the selectable dataset properties, operator is one of = (equal), != (not equal), > (larger) , >= (larger or equal), < (smaller), <=(smaller or equal) and the value is either a single word or a range (two words separated by a colon `value1:value2`[1]). The selectable properties are specific to each dataset

Example: `ele=C    rnum=1,4,7,8    anum=1:1000    bfac<30`

- *within*: this selection is applied when the operator is <>. The property is a distance R in Å and the value can be either a coordinate (in the form `{x,y,z}`) or an object (in the form `.object` (for own objects) or `.dataset.object`). For the coordinate, the selection is true if the element is within a sphere of radius R around point `{x,y,z}`. For the object, the selection is true if the element is within distance R from any part of the object[2].

```
Example: 6<>{12.0,3.6,-39.7}
         10<>.struct.obj
```

- *object*: the PV is contains only to the value, which starts with a dot and is interpreted as an object (from the same dataset). Multiple objects can be given, separated by commas. The selection will be true if the element is already present in that object. (Under special circumstances, a value in the form `.dataset.object` can be given, see attachments below).

Following is a collection of selection statements to clarify the sytnax:

```
.struct new -sel rnum=1:100  // select residues 1 to 100
.struct new -sel not chain=C,D,E  // all except chains C,D and E
.struct new -sel (rnum=7,12,43 & !aname=C,N,O)
.scal new -sel 10<>[.struct.obj] // select around center of object
.scal new -sel 10<>.struct.obj   // select around limits of object
.surf new -sel .a,.b,.c       // effectively merges three objects
```

5.1.5 Modifying Objects

An object can be modified with the commands `renew` and `set`. Former carries similar functionality as the dataset command `new`, expect the name and type are fixed, and the selection and object properties will default to current selection and properties of the object if not explicetly specified.

Syntax: `.dataset.object renew [-selection SEL] [-set S]`

Calling `renew` without any parameter re-creates the object, which is sometimes useful, e.g. after the dataset restriction changes (see below).

The command `set` can be used on datasets and objects to modify properties with a comma seperated list of PVs, either in the context of `renew` or by itself:

---

1. In previous DINO versions, the range could also be expressed as <value1,value2>. **This old syntax is no longer supported!**
2. To be more precise, within distance R from any element in the object, e.g. for a structure object the atoms, a surface object the vertices etc...

```
Syntax: .dataset.object set PV1[,PV2 ...] [-selection SEL]
                                          [-range RANGE]
```

It can optionally carry the flags `-selection` and/or `-range`. If a selection is used, only the properties of the matching elements will be modified:

```
Example: // color all carbon atoms of this structure object in cyan
         .struct.all set color=cyan -sel aname=C*
```

The option `-range` allows to linearly map a property of the same or another dataset to an object property (usually color). Up to three properties for `-range` need to be specified: `src`, `prop` and `val`.

```
Syntax: .ds.obj set color=color1:color2
                   -range [src=SRC][,prop=PROP][,val=VAL1:VAL2]
```

The range properties have the following meaning:

`src`: The source dataset where to get the property from. If this is omitted or just a single dot, then the dataset of the object is used.

`prop`: The property from the dataset which is used for mapping, this is specific to each dataset (see below). If omitted, the default mappable property for the specific dataset is used.

`val`: A range which is to be mapped. Values outside this range are ignored. If omitted, the value range used is undefined.

In the tables below the R column denotes a property that can be used for applying a range: the structure dataset for instance allows `bfac`, so to color a structure object according to b-factor one would use

```
Example: .strc.obj set color=blue:red -range prop=bfac,val=0:80
         // note that 'src' is not needed
```

5.1.6 Graphical Appearance

The way an object is displayed in the graphics window is managed with the commands `render` and `material`.

```
Syntax: .ds.obj render PV[,PV]
        .ds.obj material [amb=COL],[diff=COL],
                         [spec=COL],[shin=N],[EMM=COL]
```

The render properties are explained in more detail in the individual dataset sections below.

The material properties characterise how the objects interact with the light sources. `COL` is either a full `{r,g,b}` triplet or a single (grey) value. `N` is a number between 0 and 128.

`amb`: defines how much of the ambient lightsource component is reflected from the material; ambient reflection is independent of the position of the light

`diff`: defines how much of the diffuse lightsource component is reflected from the material; diffuse reflection is dependent on the position of the light

`spec`: defines how much of the specular lightsource component is reflected from the material; specular reflection produces highlights

`shin`: shininess component, determines the brightness and size of the highlight, the higher the number, the brighter and narrower the highlight.

`emm`: simulates light emmision, but does not act as a lightsource

For more information concering material properties and the interaction with the lightsources, consult an OpenGL reference (e.g. *OpenGL Programming Manual 3rd Ed* pp201).

5.1.7 Dataset Restriction

Elements in the dataset can be restricted, so that they are not taken into account during subsequent commands.

```
Syntax: .dataset restrict SELECTION
```

All elements that do NOT match the selection are excluded from the dataset. To unrestrict the dataset, call restrict with * as selection.

```
Example: .myo restrict not rname=HEM // this residue is excluded
         .myo restrict *             // restriction is off
```

A restriction in one dataset can have an effect on other datasets as well (see e.g. attachment below). The default is an unrestricted dataset.

5.1.8 Transformation

Each dataset can be transformed individually with a rotation and a translation inside the global coordinate system. There are the following ways to transform:

- Set the properties `rot` and `trans` for absolute transformation

```
Syntax: .dataset set rot={{a,b,c},{d,e,f},{g,h,i}},trans={x,y,z}
```

- Use the commands `rotx`, `roty`, `rotz` resp. `transx`, `transy`, `transz` for relative transformation

```
Syntax: .dataset rotN angle
        .dataset transN value
```

- Grab an input device

```
Syntax: .dataset grab INPUTDEVICE
```

`INPUTDEVICE` is one of `mouse`, `mouse2`, `dials`, `dials2`, `spaceball`, `spaceball2` (see “4.10 Input” on page 20).

To reset or fix the transformation:

```
Syntax: .dataset reset
        .dataset fix
```

`reset` will set the transformation back to identity. `fix` will apply the transformation to the dataset elements and then set the transformation back to identity.

5.2 Database Manager

There are two ways to create a dataset: loading a file or issuing the `new` command. Currently, the geometric primitives dataset can only be created with `new` and all others (structure, scalar field, surface, topography) must be loaded from a file.

5.2.1 Loading Datasets

```
Syntax: load file.ext [-type T] [-name N] [-swap]
```

The type is guessed from the extension (see Table 7 (P. 26)) or can be explicitly set with the `-type` flag. Files can be gzipped, this will be recognized and the file gunzipped on the fly (adding the `.gz` extension to `file.ext` is not necessary). If no `name` is given, the filename without extension will be used. The option `-swap` swaps the binary values from LSB to MSB or vice versa. Use this when exchanging binary files between platforms that have a different byte order (e.g. SGI and Linux). See

Table 7: Database Manager supported file formats

extensions	type	description	dataset	format
`.pdb .ent`	`pdb`	Brookhaven / RCSB PDB coordinates	structure	ASCII
`.xpl`	`xplorc`	XPLOR 3.x coordinate file	structure	ASCII
`.cnsc`	`cnsc`	CNS 0.9 coordinate file	structure	ASCII
`.crd`	`charmm`	CHARMM coordinate file	structure	ASCII
`.pqr`	`pqr`	MEAD coordinate file	structure	ASCII
`.map`	`ccp4`	CCP4 electron density and mask	scalar field	binary
`.xmp .xmap`	`xplorb`	XPLOR 3.x binary electron density map	scalar field	binary
`.cmp .cmap`	`cnsb`	CNS 0.9 binary electron density map	scalar field	binary
`.uhb .uhbd .pot`	`uhbd`	UHBD potential	scalar field	binary
`.fld .mead`	`mead`	MEAD field	scalar field	binary
`.grd .ins`	`delphi`	DELPHI electrostatic potential	scalar field	binary
	`xplora`	XPLOR 3.x ASCII map	scalar field	ASCII
	`cnsa`	CNS 0.9 ASCII map	scalar field	ASCII
	`delphig`	DELPHI-like 64^3 fixed map (i.e. from GRASP)	scalar field	binary
`.face + .vert`	`msms`	MSMS output	surface	ASCII
`.vet .msp`	`msp`	MSP output	surface	ASCII
`.tiff`	`topo`	square greyscale TIFF image	topography	n.a.

Table 7 (P. 26) for all supported formats.

5.2.2 Creating Datasets

In future versions, all datasets can be created from scratch, but at the moment this is restricted to the geometric primitives dataset:

```
Syntax: new TYPE [-name N]
```

If the name is not given, the dataset type will be used.

```
Example: new geom -name g
```

5.2.3 Other commands

A dataset can be removed with `delete` and renamed with `rename`. *Note* that there is no dot in front of the dataset names:

```
Syntax: delete dataset
        rename old new
```

`list` will display all currently loaded datasets.

6 Structure Dataset

The structure dataset is probably the most complex of all datasets present in DINO. Its most basic element is the atom. There are two different ways to think about the organization of atoms in this dataset. In one organization, all atoms are in a list and carry properties such as position, name, number, chemical-info (element, charge, vdW-radius), residue-name, residue-number, chain-id, model etc .... Another way is to envision a hierarchy where the structure dataset consists of of or more models, each model consists of one or more chains, each chain consists of one or more residues and each residue is build up from atoms, which carry atom-only properties such as position, chemical-info, name and number.

Most file-formats use the first representation. DINO internaly uses the second way to organize the structural data, but for selection and direct addressing purposes, its easier to think about it in the first way.

Additionally, a trajectory can be loaded into a structure dataset. A trajectory consists of a collection of frames, each with the same set of atoms but different coordinates for each atom (see also below). The trajectory can be played while retaining full interactivity.

Table 8: Structure Dataset and Object Properties

Property	S[a]	G[b]	Description
Dataset Properties			
center	X	X	center of rotation, default equals center of gravity
rot	X	X	rotation matrix
trans	X	X	translation vector
smode	X	X	selection mode, either `atom` or `residue`, default is `atom`
tfast	X	X	fast updates of trajectories, cannot be used with render mode custom, default is on
Object Properties			
center		X	geometric center of object

a. indicates that this property can be modified with set
b. indicates that this property can be retrieved with get

6.1 Structure Objects

New structure objects are created with `new`. There are two object types for the structure dataset:

- `connect`: Connects atoms with bonds using the chemical connectivity, this is the default object type, all selection properties are considered.
- `trace`: The central atoms of each residue (`CA` for proteins, `C3'` for nucleic acids) are connected to each other in their sequential order according to residue-numbers. A discontinuity in the residue numbering or a change in chain identifier causes a break in the trace. The selection properties `aname`, `anum` and `ele` have no effect.

```
Syntax: .struct new [-name NAME][-type TYPE][-set SET]
                    [-selection SEL]
```

Table 9: Structure Dataset Element Properties

Property	S[a]	G[b]	L[c]	R[d]	Description
element properties shared between dataset and object; modified with `.dataset set`					
`aname`		X	X		atom name as present in coordinate file
`anum`		X	X	X	atom number as present in coordinate file
`bfac`		X	X	X	crystallographic temperature factor (if present)
`chain`		X	X		alphanumeric chain id
`class`		X			class of structure: `protein`, `na` or `misc`
`ele`		X	X		chemical element symbol
`model`		X	X	X	model number
`rname`		X	X		residue name as present in coordinate file
`rnum`		X	X	X	residue number
`rtype`	X	X	X		residue type: `coil` (default), `helix` or `strand`
`weight` `occ`		X	X		weight or crystallographic occupancy (if present)
`xyz`		X			position of atom
`x`			X	X	x-coordinate of atom
`y`			X	X	y-coordinate of atom
`z`			X	X	z-coordinate of atom
element properties copied from dataset to object, setting on the dataset changes default					
`color`	X	X			color, either a color name or an explicit `{r,g,b}` triplet
`vdwr`	X	X	X	X	VdWaals radius

a. indicates wether this property can bet changed with `set`
b. indicates wether this property can be retrieved with `get`
c. indicates a selectable property
d. indicates a property that can be used in a `range` statement

`NAME` defaults to the dataset name. `TYPE` defaults to `connect`. Object properties that can be set are listed in Table 8 (P. 27). A selection with selectable atom properties (column L in Table 9 (P. 28)) can be given.

```
Example: load myo.pdb
         .myo new -name all
         .myo new -name ca -type trace
         .myo new -name prot -sel not rname=HEM
         .myo new -name helix1 -type trace -sel rnum=4:14
```

6.2 Structure Dataset Commands

Following are all structure dataset commands:

`new`: creates a new object (see “6.1 Structure Objects” on page 27)

`set get`: retrieves or modifies a dataset property (columns S and G in Table 8 (P. 27))

```
Syntax: .struct set PV[, PV ...]
        .struct get P
```

`restrict`: shadow out a subset of the dataset which will not be considered in selection or property setting (see “5.1.7 Dataset Restriction” on page 25).

```
Syntax: .struct restrict SELECTION
```

`delete`: remove one or more objects

```
Syntax: .struct delete obj [obj2 ...]
```

`load step play stop`: trajectory related commands (see “6.6 Trajectories” on page 31)

`grab reset fix center`: transformation related commands (see “5.1.8 Transformation” on page 25); `reset` additionally can have the parameters `all` (default) or `center`, the latter will set the center of rotation back to the center of geometry after a modification with `scene set center`

```
Syntax: .struct reset [all | center]
```

```
Example: .struct grab dials          // dial input moves dataset
         .struct set center=[$CS]
         // rotation now around current selection
         .struct reset center
         // rotation now around geometric center of dataset
         scene grab dials          // dial input back to camera
```

`write`: writes the dataset into a file (*taking restriction into account*!)

```
Syntax: .struct write file.ext [-type T]
```

The type of the file is recognized by its extension if not explicitly given. Supported types with extensions are listed in Table 10 (P. 29).

`connect`: Adds a bond between two atoms (see “6.5 Connectivity” on page 31).

Table 10: Structure Dataset Write formats

extension	type	description
`.pdb`	`pdb`	RCSB / Brookhaven PDB format
`.xpl`	`xplorc`	XPLOR / CNS format
`.crd`	`charmm`	CHARMM format
`.xyzr`	`xyzr`	suitable as input for MSMS

6.3 Structure Object Commands

Structure object commands are issued as:

```
Syntax: .struct.obj command [parameters]
```

Following is a list of all structure object commands:

`renew`: Similar to dataset command `new`, except the type and name are already determined, so only selection and setting is possible. If either selection or setting (or both) are omitted, the current selection resp. setting is maintained (see “5.1.5 Modifying Objects” on page 23).

```
Syntax: .struct.obj renew [-selection SEL] [-set SET]
```

`show hide`: turns display of the object in the graphics window on or off. Identical to clicking on the object name in the object menu (see “4.12 The GUI” on page 20).

```
Syntax: .struct.obj show
        .struct.obj hide
```

`clear`: removes all labels. Hint: use `.struct.* clear` to get rid of all labels.

`set get`: modify or retrieve structure object properties (columns S and G in Table 8 (P. 27) under structure object properties and see “5.1.5 Modifying Objects” on page 23).

```
Syntax: .struct.obj get P
        .struct.obj set PV [-selection SEL]
                  [-range [src=SRC][,PROP=prop][val=VAL1:VAL2]]
```

```
Example: .myo.all set color=cyan -sel ele=C
         .myo.ca set color=green:purple -range prop=rnum,val=1:146
```

`render`: modifies the graphical appearance of an object (see “6.7 Render & Graphical Appearance” on page 32)

`write`: Similar to the the dataset command `write`, the same file-types are supported (see Table 10 (P. 29)). Only the atoms in the object are written and the dataset restriction is ignored.

```
Syntax: .struct.obj write file.ext [-type T]
```

If no command is given, then `get center` is used. This is an useful shortcut for e.g. centering the object in the graphics window:

```
Example: // these two are identical
         scene center [.myo.all get center]
         scene center [.myo.all]
```

6.4 Structure Element Commands

The syntax for addressing an element in a structure dataset is

```
Syntax: .struct:ELEMENT command [parameters]
```

The format of ELEMENT depends on the dataset itself, to be more precise whether chain and/or model are present. The general format is

`ELEMENT = [model.][chain.][rnum.]aname`

`model`, `chain` and `rnum` are only required if the structure dataset contains different models, chains and/or residues. The atom name is case sensitive and must be entered as present in the coordinate file.

```
Example: // no chain or model
         .struct:143.CA
         // chain also present
         .struct:B.99.N
         // model and chain present
         .struct:3.A.76.O
         // model but no chain
         .struct:2.81.CB
```

Since the dataset 'knows' if a chain or model is present, it expects a certain format. No wild-cards can be used.

The only command implemented so far is get, which retrieves an atom property:

```
Syntax: .struct:[model.][chain.][rnum.]aname get PROPERTY
```

If no command is given, `get xyz` (the coordinate of the atom) is assumed.

```
Example: scene center [.myo:148.FE]
         // is equal to [.myo:148.FE get xyz]
```

6.5 Connectivity

auto-connectivity

As mentioned above, a structure dataset is automatically created when loading a file which contains coordinate-data. Although most coordinate-files do not contain explicit connectivity information, the connectivity is implicitly given by:

- The relationships between atoms as defined by chain-id, residue-number, residue-name and atom-name
- The distance between two atoms, which is different for bonded atoms and atoms in vdW contact.

DINO uses an auto-connectivity routine that exploits both implicit and explicit connectivity rules. It contains an internal connectivity table for the 20 amino-acids and the 5 bases, including a rule which atoms connect residues n and n+1[1] . If an atom is encountered that is not defined by the internal connectivity table, it is connected to all atoms that lie within CUTOFF of its position; where CUTOFF is defined as the sum of the two van der Waals radii divided by two. Additionally, connectivity defined in the coordinate-file is considered (e.g. CONECT in PDB). As a consequence, disulfide bridges must be either explicitely defined with CONECT, or the atom name of the cystein sulfur (normally SG) must be changed to force distance-based connectivity (e.g. into SS).

manual connectivity

The structure dataset command `connect` can be utilized to connect two atoms together

```
Syntax: .struct connect ELEMENT1 ELEMENT2
```

`ELEMENT` is a structure dataset element in the form as explained in the direct access section above.

```
Example: .struct connect A.13.SG B.156.SG
```

6.6 Trajectories

Trajectories are an addition to a structure dataset. They provide a collection of frames which contain new coordinates for each atom of the structure dataset. First a trajectory needs to be loaded from a file

```
Syntax: .struct load file.ext [-type T] [-swap]
```

If the type is not given, it is guessed from the extension. Currently, CHARMM (extension `.trj` type `charmm`) and CNS (extension `.crd`, type `cns`) trajectories are supported The parameter `-swap` can be used to apply byte-order swapping to the trajectory.

Once trajectories are loaded, they can be played continuously or each frame can be looked at one by one.

```
Syntax: .struct play [-wait w] [-begin b] [-end e]
                     [-step s] [-mode m]
        .struct stop
        .struct step [n]
```

1. The structure of this internal table is quite simple.In a future version of DINO, users will be able to supply their own connectivity for custom residues.

`wait` denotes the number of wait-cycles until the next frame is displayed, default is 0. `begin` and `end` specify from which to which frame the trajectory shall run, default is 1 to max. `mode` can be `loop` (default) or `rock`.

While the trajectory is playing, full interactivity is available.

The structure dataset property `tfast` can be used to turn off the recalculation of internal parameters needed for render mode custom. This will considerably speed up trajectory display, but will only work if all structure dataset objects are not in render mode `custom`. This flag is true (fast trajectory) per default.

6.7 Render & Graphical Appearance

The command `render` modifies the graphical appearance of an object:

Syntax: `.struct.obj render PV [,PV ...]`

The render properties applicable to the structure dataset objects are given in Table 11 (P. 33). Depending on the type of object (connect or trace), several rendering modes are available, specified as

Syntax: `.struct.obj render MODE`

Modes:

- `simple` (default mode for `connect` and `trace`): bonds are drawn as lines with width `linew`, each half colored according to the color of the connecting atom.
- `custom` (`connect` and `trace`): bonds are drawn as cylinders with width `bw`, atoms are drawn as spheres with radius `sr`
- `cpk` (`connect`): atoms are displayed as spheres with a radius of their `vdwr`
- `tube` (`trace`): backbone is rendered as spline tube of radius `bw` and axial ratio of `tuber`
- `hsc` (`trace`):secondary structure rendering of protein or dna trace; a continuous spline with interpolation runs exactly through the atom positions.
  protein rendering: `coils` as tubes of radius `bw` and axial ratio of `tuber`, `helices` with width `helixw` and thickness `helixt`, and strands with width `strandw` and thickness `strandt`
  dna rendering: backbone as tubes of radius `bw` and axial ratio of `tuber`, purines and pyrimidines are cartoonlike displayed with approximate tilting of the sugar and the base
- `sline` (`trace`): continuous spline displayed as line with linewidth `lw`

Additionally, the command `material` can be used to change the material settings, this has been described above, see “5.1.6 Graphical Appearance” on page 24.

Table 11: Structure Dataset Object Render Properties

Property	C	T	Description	Default
`arrowt`		X	width of arrow of strand in hsc rendering, expressed as ratio to `strandw`	0.5
`bw`	X	X	bondwidth, affects modes `custom`, `tube` and `hsc`	0.2
`detail`		X	number of spline interpolations for modes hsc, tube and sline, increase for better quality	5
`detail2`	X	X	number of spherical interpolations for spheres and circular subdivisions (per 90deg) for modes `custom`, `hsc` and `tube`	3
`fast`	X	X	turns off point and line antialiasing to increase drawing speed	
`helixt`		X	helix thickness in mode `hsc`	0.3
`helixw`		X	helix width in mode `hsc`	1.0
`intpol`		X	if this flag is true, colors will be smoothly interpolated on segments for render modes `sline`, `tube` and `hsc`	true
`lw`	X	X	linewidth, will accept fractional values, but effect depends on hardware implementation	1.0
`nice`	X	X	point and line antialiasing is active	X
`sr`	X		sphere radius in custom mode	0.2
`strandm`		X	strand method in mode `hsc`, 0 is default, 1 causes smoothing	0
`strandt`		X	strand thickness in mode `hsc`	0.3
`strandw`		X	strand width in mode `hsc`	1.2
`t`	X	X	transparency, 1.0 is opaque, 0.0 fully transparent; works on lines in modes `simple` and `sline`; works also for modes `hsc` and `tube`	1.0
`tuber`		X	axial ratio of tube cross section	1.0

7 Surface Dataset

The surface dataset contains information about a molecular surface calculated with an external program, such as MSMS[1] or MSP[2]. It is usually generated by rolling a solvent sphere of a fixed radius around the structure of interest, producing (if possible equidistant) points on the surface which are connected to triangles, approximating the real surface. The spacing of the points is usually adjustable to give the desired resolution. Each point (or vertex) additionally carries a normal vector which is utilized during visualization to smooth the surface, and it appears less coarse than an identical surface with triangular normalization.

The power of the surface dataset lies in its attachment feature: each vertex can be attached to an atom of a structure dataset and inherits that atom's properties (such as name, number, residue, chain, etc ...).

Table 12: Surface Dataset and Object Properties

Property	S[a]	G[b]	Description
Dataset Properties			
`center`		X	geometric center of dataset
`rot`	X	X	rotation matrix
`trans`	X	X	translation vector
`smode`	X	X	selection mode, either `all` (default) or `any`
Object Properties			
`center`		X	geometric center of object

a. indicates that this property can be modified with set
b. indicates that this property can be retrieved with get

7.1 Attachments

```
Syntax: .surf attach .struct [-cutoff R]
        .surf attach none | off
```

This command will attach the surface dataset `surf` to the structure dataset `struct`: each vertex of surf is assigned the closest atom of struct - if this atom lies within the cutoff radius R (default 3.0). As an additional criteria, only atoms that pass the restriction of `struct` are taken into account (see “5.1.7 Dataset Restriction” on page 25). Vertices that are not assigned an atom are left unchanged, i.e. a previous attachment is not removed. In this way multiple attachments can be issued, possibly with different structure datasets.

To clear all attachment information, attach is called with `none` or `off`.

As a result of an attachment, each surface vertex additionally carries the information of a single atom. Atomic properties (as listed in Table 9 (P. 28)) can be used during selection statements involving surfaces (e.g. during `new`, `renew` and `set`).

```
Example: // see example in next section
```

1. http://www.scripps.edu/pub/olson-web/people/sanner/html/msms_home.html
2. http://www.biohedron.com

7.2 Surface Objects

Surface dataset objects are created with the command new:

Syntax: `.surf new [-name NAME] [-selection SEL] [-set SET]`

There is currently only one object type for the surface dataset, so the type does not need to be specified. NAME defaults to the name of the surface dataset. SEL contains a selection expression, which can contain selectable atom properties if the surface is attached to at least one structure dataset (see Table 9 (P. 28)). Object properties can be defined with -set (see Table 12 (P. 34)).

Example:
```
// example with myo.pdb and myo.face / myo.vert
load myo.pdb -name struct
load myo -type msms -name surf
.surf attach .struct
.surf new -name s1 // new object
// color surface that lies closest to HEM group
.surf.s1 set color=green -sel rname=HEM
.surf attach off // reset attachment
.struct restrict rname=HEM
.surf attach .struct // this time restriction is present
.surf new -name s2 // new object
// same command as above !
.surf.s2 set color=green -sel rname=HEM
// compare surface coloring of s1 and s2 ...
```

Hint: to select only the part of the surface that carries an attachment, force the selection to evaluate an atom property which is always true.

Example: `.surf.all set color=yellow -sel aname=*`

Table 13: Surface Dataset Element Properties

Property	S[a]	G[b]	L[c]	R[d]	Description
element properties shared between dataset and object					
none, unless attached to structure dataset, then all properties from Table 9 (P. 28), but only columns `L` & `R`					
element properties copied from dataset to object, setting on dataset changes default					
`color`	X	X			Surface Color

a. indicates wether this property can be changed with set
b. indicates wether this property can be retrieved with get
c. indicates a selectable property
d. indicates a property that can be used in a range statement

7.3 Surface Dataset Commands

Following are all surface dataset commands:

`new`: create a new object (see “7.2 Surface Objects” on page 35).

`set get`: retrieve or modify a dataset property (see Table 12 (P. 34))

Syntax:
```
.surf set PV[, PV ...]
.surf get P
```

`restrict`: shadow out a subset of the dataset which will not be considered in selection or property setting (see “5.1.7 Dataset Restriction” on page 25).

```
Syntax: .surf restrict SELECTION
```

`delete`: remove one or more objects

```
Syntax: .surf delete obj [obj2 ...]
```

`attach`: attach surface to a structure dataset (see “7.1 Attachments” on page 34).
`grab reset fix center`: transformation related commands (see “5.1.8 Transformation” on page 25).

7.4 Surface Object Commands

Surface object commands are issued as

```
Syntax: .surf.obj command [parameters]
```

Following is a list of all surface object commands:

`renew`: Similar to new command, except the type and name are already determined, so only selection and setting is possible. If either selection or setting (or both) are omitted, the current selection resp. setting is maintained (see “5.1.5 Modifying Objects” on page 23).

```
Syntax: .surf.obj renew [-selection SEL] [-set SET]
```

`show hide`: turns display of the object in the graphics window on or off. Identical to clicking on the object name in the object menu (see “4.12 The GUI” on page 20).

```
Syntax: .surf.obj show
        .surf.obj hide
```

`set get`: modify or retrieve surface object properties (See Table 12 (P. 34) and see “5.1.5 Modifying Objects” on page 23).

```
Syntax: .surf.obj get P
        .surf.obj set PV [-selection SEL]
                     [-range [src=SRC][,PROP=prop][val=VAL1:VAL2]]
```

```
Example: .surf attach .myo; .surf new -name all
         .surf.all set color=green:blue -range prop=rnum,val=1:146
```

`render`: modifies the graphical appearance of an object, see below.

7.5 Render & Graphical Appearance

The command render changes the graphical appearance of a surface dataset object:

```
Syntax: .surf.obj render PV [,PV ...]
```

The render properties applicable are listed in Table 14 (P. 37). Several rendering modes are available, specified as

```
Syntax: .surf.obj render MODE
```

modes:

- `dot`: vertices are displayed as points only
- `line`: faces are outlined
- `fill`: default mode, faces are filled and lit

Table 14: Surface Dataset Render Properties

Property	Description	Default
`t`	transparency, 1.0 is opaque, 0.0 is fully transparent	1.0
`light1`	lighting is only applied to the front face, leaving the backside practically unlit and dark	X
`light2`	lighting is applied to both sides of the surface	

Material parameters can be modified with the command `material` (see “5.1.6 Graphical Appearance” on page 24).

8 Scalar Field Dataset

The scalar field dataset is a 3-dimensional grid that contains a single scalar value at each grid point. To visualize it fully would require a 4-dimensional representation, which is currently beyond the capabilities of a computer workstation One interpretation is called *iso-contouring*, which creates an iso-surface from the scalar field with each point on the surface (ideally) having the same scalar value. Another interpretation is to draw each grid-point with its scalar value represented as a point of varying thickness, or to blend the grid points as a texture (as exploited in medical resonance imaging, also called volume rendering).

The grid is regular, but needs not be cubic. It can consist of arbitrary axis a, b and c with arbitrary angles α, β and γ. Its origin (grid point 0,0,0) will be mapped to some point in cartesian space (offset).

Probably the three most common uses for scalar fields in structural biology are electron density from x-ray data, 3D reconstructions from electron microscopy and electrostatic potential maps from theoretical calculations.

Table 15: Scalar Field Dataset and Object Properties

Property	S[a]	G[b]	Description
Dataset Properties			
`center`		X	geometric center of dataset
`rot`	X	X	rotation matrix, default is identity
`trans`	X	X	translation vector, default is identity
`edge`	X	X	value used beyond defined grid-point, default is 0.0
`scale`	X	X	scalar factor that is multiplied with every coordinate prior to object creation, existing objects must be renewed, default is 1.0
Object Properties			
`center`	X	X	center of object, default is center of dataset
`size`	X	X	size of object in grid-units; can be either a scalar (cubic size) or in form of {xs,ys,zs}, default is 30
`level`	X	X	(object type `contour`) contour level, if `s` is appended then value is interpreted as sigma (standard deviation) units, default is 0.0
`step`	X	X	step-size along `grid` during object creation, default is 1
`color`	X		(object type `contour`) uniform color for object, default is white

a. indicates that this property can be modified with `set`
b. indicates that this property can be modified with `get`

8.1 Scalar Field Objects

A scalar field object is created with the command `new`. In contrast to the structure and surface dataset, the `-set` parameter is quite important, while the selection is not often employed.

Syntax: `.scal new [-name NAME] [-type T][-selection SEL][-set SET]`

NAME defaults to the name of the dataset. SET can be any of the settables object properties (see Table 15 (P. 38)). There are two scalar field object types:

- `contour`: creates an iso-surface around `center`, with dimensions `size`, at the scalar value `level`, using a step-size of `step`. This is the default type.
- `grid`: creates a grid around `center`, with dimensions `size`, using a step-size of `step`, the resulting points are displayed with radius `radius`

```
Example: .scal new -name m -set center=[.struct], size=40,
                                  level=1.5s, color=blue
```

```
Example: .scal new -name g -type grid -set size=40
```

Although a selection can already be given during object creation, its maybe easier to call a renew (to keep the lines short):

```
Example: .scal.m renew -sel 10<>.struct.inhib
```

All scalar field object properties except color will not take immediate effect, only when a renew (see below) is called. This is usually combined:

```
Example: .map new -name m -set center=[.map],size=40,level=1.0s
         // change the size
         .map.m renew -set size=50
         // change the level
         .map.m renew -set level=1.2s
```

Table 16: Scalar Field Dataset Element Properties

property	S[a]	G[b]	L[c]	R[d]	Description
`color`	X				(for object type `grid`) color of individual grid point
`radius`	X				(for object type `grid`) radius of individual grid point
`v`			X	X	absolut value

a. indicates that this property can be modified with set
b. indicates that this property can be modified with get
c. indicates a selectable property
d. inidicates a property that can be used in a range statement

8.2 Scalar Field Dataset Commands

Following are all scalar field dataset commands:

`new`: create a new object (see “8.1 Scalar Field Objects” on page 38).

`set get`: retrieve or modify a dataset property (Table 15 (P. 38)).

```
Syntax: .scal set PV[, PV ...]
        .scal get P
```

`restrict`: shadow out a subset of the dataset which will not be considered in selection or property setting (see “5.1.7 Dataset Restriction” on page 25).

```
Syntax: .scal restrict SELECTION
```

`delete`: remove one or more objects.

```
Syntax: .scal delete obj [obj2 ...]
```

`grab reset fix center`: transformation related commands (see “5.1.8 Transformation” on page 25).

8.3 Scalar Field Object Commands

Commands directed against a scalar field object are issued as

```
Syntax: .scal.obj command [parameters]
```

Following is a list of all scalar field object commands:

`renew`: Similar to new command, except the type and name are already determined, so only selection and setting is possible. If either selection or setting (or both) are omitted, the current selection resp. setting is maintained (see “5.1.5 Modifying Objects” on page 23).

```
Syntax: .scal.obj renew [-selection SEL] [-set SET]
```

```
Example: .map.m renew -set level=1.2s, size=50, color=blue
```

`show hide`: turns display of the object in the graphics window on or off. Identical to clicking on the object name in the object menu (see “4.12 The GUI” on page 20)

```
Syntax: .scal.obj show
        .scal.obj hide
```

`set get`: modify or retrieve scalar field object properties (See Table 15 (P. 38) and see “5.1.5 Modifying Objects” on page 23)

```
Syntax: .scal.obj get P
        .scal.obj set PV [-selection SEL]
                     [-range [src=SRC][,PROP=prop][val=VAL1:VAL2]]
```

`render`: modifies the graphical appearance of an object (see “8.4 Render & Graphical Appearance” on page 40)

8.4 Render & Graphical Appearance

The command render modifies the graphical appearance of a scalar field dataset object:

```
Syntax: .scal.obj render PV [,PV ...]
```

The render properties that can be used for the scalar field dataset objects are given in Table 17 (P. 41). Depending on the object type (`contour` or `grid`), different rendering modes are available, specified as:

```
Syntax: .scal.obj render MODE
```

where mode is one of:

- `dot`: (for type `contour`) vertices are displayed as points only
- `line`: (for type `contour`) default mode, faces are outlined
- `fill`: (for type `contour`) faces are filled and lit
- `on` / `off`: (for type `grid`) turns rendering on (spheres) or off (points)

Material parameters can be modified with the command `material` (see “5.1.6 Graphical Appearance” on page 24).

8.5 Mapping Scalar Fields to Surfaces

DINO provides an easy way to map scalar field values to a surface. This can be accomplished by using the surface object command `set` with `-range` and as `src` the scalar field. Following is a generic example, assuming a surface named `surf` and an electrostatic potential as a scalar field dataset called `pot`

```
Example: // load surf ...
         // load pot ...
         .surf new -name all              // generate surface object
         .surf.all set color=red:white -range src=.pot, val=-5:0
         .surf.all set color=red:red   -range src=.pot, val=-999:-5
         .surf.all set color=white:blue -range src=.pot, val= 0:5
         .surf.all set color=blue:blue -range src=.pot, val=5::999
```

Table 17: Scalar Field Dataset Object Render Properties

Property	C[a]	G[b]	Description	Default
`fast`	X	X	turns off point and line antialiasing to increase drawing speed	
`light1`	X		lighting is only applied to the front face, leaving the backside practically unlit and dark	
`light2`	X		lighting is applied to both sides of the surface	X
`lw`	X		linewidth	1.0
`nice`	X	X	turns on point and line antialiasing	X
`ps`	X	X	pointsize	1.0
`t`	X		transparency, 1.0 is opaque, 0.0 is fully transparent	1.0

a. applicable to object type contour
b. applicable to object type grid

9 Topography Dataset

A surface topograph is a 2-dimensional grid that contains a scalar height value at each grid-point. In contrast to the scalar field, which would require 4 dimensions to fully visualize it (see above), the surface topograph can be completely visualized in 3 dimensions by constructing a 3-dimensional surface from the grid and the height-values.

Table 18: Topography Dataset and Object Properties

Property	S[a]	G[b]	Description
Dataset Properties			
`center`		X	geometric center of dataset
`scalexy`	X	X	scaling factor of xy plane, default is 1.0
`scalez`	X	X	scaling factor of z-height, default is 1.0
`trans`	X	X	translation vector
`rot`	X	X	rotation matrix
Object Properties			
`center`		X	geometric center of object
`step`	X	X	step-size along grid during object creation
`lstart`	X	X	relativ contour level start, value between 0.0 and 1.0
`lend`	X	X	relativ contour level end, value between 0.0 and 1.0
`lstep`	X	X	relativ contour level step, value between 0.0 and 1.0

a. indicates that this property can be modified with `set`
b. indicates that this property can be retrieved with `get`

9.1 Attachments

Attachments are not yet implemented for the topography dataset.

9.2 Topography Objects

Two topography object types are implemented:

- `surface`: A triangulated surface is constructed from the dataset. From 4 neighboring grid-points - `{u,v} {u+step,v} {u,v+step} {u+step,v+step}` - a middle point is interpolated and 4 triangles are formed, resulting in a surface.
- `contour`: Contour lines are generated from the dataset and are height-scaled, starting at `lstart`, ending at `lend`, contouring every `lstep`

A new object is created with the command `new`:

Syntax: `.topo new [-name NAME] [-type TYPE] [-set SET] [-sel SEL]`

If no name is given, the dataset-name is used. TYPE defaults to `surface`. SET can be a list with PVs (Table 18 (P. 42)) and SEL can include all selecteable element properties (see Table 19 (P. 43)).

Table 19: Topography Dataset Element Properties

property	S[a]	G[b]	L[c]	R[d]	Description
`color`	X				color of element
`u`			X	X	u-value of original topograph
`v`			X	X	v-value of original topograph
`h`			X	X	height

a. property can be modified with set
b. property can be retrieved with set
c. selecteable property
d. property can be used in range statement

9.3 Topography Dataset Commands

Following are all topography dataset commands:

`new`: create a new object (see “9.2 Topography Objects” on page 42)

`set get`: retrieve or modify a dataset property (Table 18 (P. 42))

```
Syntax: .topo set PV[, PV ...]
        .topo get P
```

`restrict`: shadow out a subset of the dataset which will not be considered in selection or property setting (see “5.1.7 Dataset Restriction” on page 25)

```
Syntax: .topo restrict SELECTION
```

`delete`: remove one or more objects

```
Syntax: .topo delete obj [obj2 ...]
```

`attach`: will attach topograph to a structure dataset (*not yet implemented*)

`grab reset fix center`: transformation related commands (see “5.1.8 Transformation” on page 25)

9.4 Topography Object Commands

Commands directed against a topography object are issued as

```
Syntax: .topo.obj command [parameters]
```

Following is a list of all topography object commands:

`renew`: Similar to new command, except the type and name are already determined, so only selection and setting is possible. If either selection or setting (or both) are omitted, the current selection resp. setting is maintained.

```
Syntax: .topo.obj renew [-selection SEL] [-set SET]
```

`show hide`: turns display of the object in the graphics window on or off. Identical to clicking on the object name in the object menu (see see “4.12 The GUI” on page 20)

```
Syntax: .topo.obj show
        .topo.obj hide
```

`set get`: modify or retrieve topo object properties (See Table 18 (P. 42) and see “5.1.5 Modifying Objects” on page 23).

```
Syntax: .topo.obj get P
        .topo.obj set PV [-selection SEL]
                  [-range [src=SRC][,PROP=prop][val=VAL1:VAL2]]
```

`render`: modifies the graphical appearance of an object (see “9.5 Render & Graphical Appearance” on page 44).

9.5 Render & Graphical Appearance

The command `render` modifies the graphical appearance of a topograph object:

```
Syntax: .topo.obj render PV [,PV ...]
```

Several rendering modes are available, specified as:

```
Syntax: .topo.obj render MODE
```

where `MODE` is one of:

- `dot`: (object type `surface`) vertices are displayed as points only
- `line`: (object type `surface`) faces are outlined
- `fill`: (object type `surface`) default mode, faces are filled and lit

Table 20: Topography Dataset Render Properties

Property	Description	Default
`t`	transparency, 1.0 is opaque, 0.0 is fully transparent	1.0
`light1`	lighting is only applied to the front face, leaving the backside practically unlit and dark	
`light2`	lighting is applied to both sides of the surface	X

9.6 Example

Following is an example using a 6-fold symmetrized correlation average of an AFM-topograph of an HPI layer[1]:

```
Example: // load a topography dataset
         load hpi.tiff
         // set scaling (in nm)
         .hpi set scalexy=33,scalez=3
         // create a new object (of default type surface)
         .hpi new -name s
         // create a new contour object
         .hpi new -name c -type contour -set lstep=0.1
         // color contour lines
         .hpi.c set color=yellow
         // color surface with range
         .hpi.s set color=red4:yellow -range prop=h,val=0:1
```

1. D.J. Muller, D. Fotiadis, A. Engel FEBS Letters 430 (1998)

10 Geometric Primitives Dataset

The geometric primitives dataset is designed to allow arbitrary geometrical objects - points, lines, triangles and rectangles - to be implemented in the scene, along with the other objects. Its concept and syntax are similar to the above datasets. This dataset is an experimental feature and its implementation is far from complete.

As mentioned above (see “5.2.2 Creating Datasets” on page 26), a geometric dataset is created with the dbm command `new`. Then, one or more objects can be generated with the dataset command `new`. The objects will at first contain nothing; the object command `add` is used to append new primitives.

```
Syntax: new geom [-name N]
        .geom new [-name N]
        .geom.obj add PRIMITIVE POSITION
```

Valid PRIMITIVEs are `point`, `line`, `tri` and `rect`. POSITION depends on the PRIMITIVE:

```
Syntax: point : p={x1,x2,x3}
        line : p={{x1,y1,z1},{x2,y2,z2}}
        tri  : p={x1,y1,z1},{x2,y2,z2},{x3,y3,z3}}
        rect : p={{x1,y1,z1},{x2,y2,z2},{x3,y3,z3},{x4,y4,z4}}
```

Properties that can be modified with `set` are `color` (`c`) and `radius` (`r`):

```
Syntax: .geom.obj set [c=COLOR] [r=RADIUS]
```

Rendering geometric objects can be modified with the object command `render`. The two modes `on` and `off` switch between sphere/dot (`point`), cylinder/line (`line`) and filled/outline (`tri` & `rect`) representation. The radius has an effect only in cylinder and sphere representation. Specific `line` render properties are `stipple`, `stipplei` and `stippleo`.

```
Syntax: .geom.obj render [on|off] [[!]stipple][,]
                         [stipplei=VAL][,]  [stippleo=VAL]]
```

`stipplei` and `stippleo` are floats that describe the length of the segments (`stipplei`) and the gaps (`stippleo`).

```
Example: // axis at origin
         new geom -name g
         .g new // name is ommited and defaults to g
         .g.g add line p={{0,0,0},{10,0,0} // x-axis
         .g.g add line p={{0,0,0},{0,10,0} // y-axis
         .g.g add line p={{0,0,0},{0,0,10} // z-axis
         .g.g set c=green,r=0.3
         .g.g render on                            // cylinder mode
         // turn stippling on
         .g.g render stipple, stipplei=0.8, stippleo=0.7
         // turn stippling off
         .g.g render !stipple
         .g.g render off                        // line mode
```

Please see the DINO homepage for examples involving the geom datatset.

11 Appendix

11.1 Changes from 0.7 to 0.8

```
fixed plenty of bugs
best visual is searched
added .struct connect
scalar-field line rendering dependent on z-orientation
added render property arrowt
added $CP
added scene property slabw
added png output
added transparency to hsc and tube rendering
added scene split
added .struct set center and .struct reset all
added structure selecteable property class
added -accum / -a keyword to output
scalar field and topograph objects have default of render light2
added insightII and grasp input format
removed RGB output
added stipple output to POVray with rounded cylinders
added comments to POVray output for stereo generation
added scene mono
added scene property detail2
added POVray output
ESC in pause mode added
added dataset commands rotx roty rotz transx transy transz
changed default variable definitions
RGB output removed
syntax for range <VAL1,VAL2> changed to VAL1:VAL2
```

11.2 Software

CCP4 homepage: `http://www.dl.ac.uk/CCP/CCP4/main.html`

CHARMM homepage: `http://yuri.harvard.edu/charmm/charmm.html`

MEAD homepage: `http://www.scripps.edu/bashford/`

MSMS homepage:
`http://www.scripps.edu/pub/olson-web/people/sanner/html/msms_home.html`

MSP homepage: `http://www.biohedron.com/`

Raster3D homepage: `http://www.bmsc.washington.edu/raster3d`

RCSB: The new PDB home: `http://www.rcsb.org/pdb/`

UHBD homepage: `http://chemcca10.ucsd.edu/~jmbriggs/uhbd.html`

X-PLOR/CNS homepage: `http://xplor.csb.yale.edu`

POVray homepage: `http://www.povray.org`

11.3 References

The C Programming Language, Second Edition, ANSI C
B.W. Kernighan and D.M. Ritchie
Prentice Hall International 1990

OpenGL Programming Guide, Third edition
J. Neider, T. Davis, M. Woo
Addison Wesley 1994

OpenGL Programming for the X Windows System
M.J. Kilgard
Addison Wesley 1996

X Windows System Programming 2nd Ed.
N. Barkakati
SAMS Publishing 1996

UNIX Systems Programming for SVR4
D.A. Curry
O'Reilly & Associates 1996

Interactive Computer Graphics
P. Burger, D. Gillies
Addison Wesley 1989

Numerical Recipes in C 2nd Ed
W.H. Press, W.T. Vetterling, S.A. Teukolsky, B.P. Flannery
Cambridge University Press 1992

PostScript Language Reference Manual (Red Book)
Adobe Systems Inc.
Addison Wesley 1987

OSF/Motif Programmers Guide (for Release 1.1)
Open Software Foundation
PTR Prentice Hall 1991

OSF/Motif Programmers Reference (for Release 1.1)
Open Software Foundation
PTR Prentice Hall 1991

Introduction to Protein Structure
C. Branden and J. Tooze
Garland 1992

Fundamentals of Crystallography
C. Giacovazzo (Ed.)
Oxford Science Publications 1992