ColorSpace Data Viewer
User manual - Version 1.0

Philippe Colantoni

1 Introduction

CDV is an OpenGL wrapper, which include high level functions. The goal of this software is not to provide a nice and smart new language but rather a fast and easy to use programming interface to visualize color data.

2 File Format

There are (actually) 3 parts in a CDV file:

1. header: This part describes only the 3D axis used and is only provided for convenience purpose. It enables users to choose standard axis (like L*a*b* color space; xyY ...) or custom axis.
2. displaylist: OpenGL can optimize 3D rendering using display list. This mechanism allows user to compile complex 3D rendering as a single display list which can be executed in the script.
3. script: This part contains all the 3D data which will be displayed.

Example

```
# this is a comment
begin header
axis RGB    # this script display data in the RGB color space
axiscolor x 1 1 1 # the R axis is white (not red anymore)
backgroundcolor 0 0 0 1 # black background
end header
begin displaylist
newlist myfirstlist # I create a new list named myfirstlist
color3f 1 0 0       # the current color is red
volume box          # display a 3D box (size 1)
endlist             # terminate the display list
end displaylist
begin script
```

```
pointsize 3         # the current point size is now 3
color3f 1 0 0       # the current color is red
begin points        # all the data provided to the graphic pipeline
                    # will be used to display points
vertex3f .5 .5 .5   # display a point at the (0.5,0.5,0.5) position
vertex3f 1 .3 .4    # display a point at the (1,.3,.4) position
end points          # no more points to visualize

pushmatrix           # we stack the current coordinate system
translatef 1 1 1     # we translate the new coordinate system
scale .3 .5 .1       # we scale the new coordinate system
calllist myfirstlist # display the red box in the new coordinate system
                     # the center of the displayed box is (1,1,1)

popmatrix            # return to the original coordinate system
end script
```

3 Software Manual

You will find in this section how to interact with CDV.

3.1 File open

There are 4 ways to open a script file:

1. using the command line;
2. by a drag and drop from a file explorer;
3. using the File/Open script menu;
4. using the File/Re-Open script menu.

When you re-open a file you keep the current 3D position and orientation.

4 CDV File Header

The header section allows user to define the axis properties (the default axis is XYZ). It is possible to build custom axis.

4.1 Debug

4.1.1 Debug OpenGL

`debugopengl`

Allows to enable the openGL debug suppont in CDV. With this headed command CDV is able to indicate all the openGL error and the shader compilation problems.

4.2 Window Properties

4.2.1 Window Size

```
windowsize width height
```

Allows user to define the size of the displayed window.

4.2.2 Window Position

```
windowposition x y
```

Allows user to define the position of the displayed window.

4.2.3 Background Color

```
backgroundcolor r g b a
```

Allows user to define the background color of the displayed window.
Example

```
begin header
  windowsize 300 500
  windowposition 100 100
end header
```

4.3 Camera Properties

4.3.1 Camera

```
camera distancetothecenter theta phi fov perspective/orthographic projection
```

Allows user to define the camera properties of the displayed window.

4.3.2 Axis properties

4.3.3 Axis

```
axis colorspace
```

or

```
axis custom nameXaxis begX endX stepsX nameYaxis begY endY stepsY nameZaxis begZ endZ stepsZ
```

Example

```
axis custom tX 0 10 10 tY 0 5 5 tZ -5 5 10
```

4.3.4 Display grid

```
displaygrid axis bool (x,y or z and true or false)
```

4.3.5 Display axis

```
displayaxis bool (true or false)
```

4.3.6 Axis color

```
axiscolor axis r g b
```

4.3.7 Hash color

```
hashcolor r g b
```

4.3.8 Bound color

```
boundcolor r g b
```

4.3.9 Axis scale factor

```
axisscalefactor axis scale
```

5 CDV Scripting Language

We propose here to describe the basic features of CDV scripting language. As we wrote previously this language is an OpenGL wrapper which include high level function (volume visualization, image mapping, display list, blending...)

5.1 Data Visualization Language

5.2 Basics

In OpenGL, most geometric objects are drawn by enclosing a series of coordinate sets that specify vertices and optionally normals, texture coordinates, and colors between glBegin/glEnd command pairs. For example, to specify a triangle with vertices at (0,0,0), (0,1,0) and (1,0,1), one could write:

```
glBegin(GL_TRIANGLES);
  glVertex3f(0,0,0);
  glVertex3f(0,1,0);
  glVertex3f(1,0,1);
glEnd();
```

Each vertex may be specified with two, three, or four coordinates four coordinates indicate a homogeneous three-dimensional location). In addition, a current normal, current texture coordinates, and current color may be used in processing each vertex. OpenGL uses normals in lighting calculations; the current normal is a three-dimensional vector that may be set by sending three coordinates that specify it. Color may consist of either red, green, blue, and alpha values (when OpenGL has been initialized to RGBA mode) or a single color index value (when initialization specified color index mode). One, two, three, or four texture coordinates determine how a texture image maps onto a primitive.

Each of the commands that specify vertex coordinates, normals, colors, or texture coordinates comes in several flavors to accommodate differing application's data formats and numbers of coordinates. Data may also be passed to these commands either as an argument list or as a pointer to a block of storage containing the data. The variants are distinguished (in the C language) by mnemonic suffixes.

Most OpenGL commands that do not specify vertices and associated information may not appear between glBegin and glEnd. This restriction allows implementations to run in an optimized mode while processing primitive specifications so that primitives may be processed as efficiently as possible.

The previous OpenGL sequence can be translated by the following script:

```
begin triangles
  vertex3f 0 0 0
  vertex3f 0 1 0
  vertex3f 1 0 1
end triangles
```

5.3 OpenGL Commands

5.4 Arrays manipulation

5.4.1 Array

```
array name type size values...
```

where name is the name of the array, type is the data type (int, float or boolean), size is the size of the array and values... the size elements of the array.

Example

```
array vertexArray float 9 0 0 0 1 0 0 1 1 1
array index int 3 0 1 2
```

5.4.2 Enableclientstate

```
enableclientstate arraystate
```

where arraystate can be: vertexarray, colorarray, normalarray or texturecoordarray.

5.4.3 Disableclientstate

```
disableclientstate arraystate
```

where arraystate can be: vertexarray, colorarray, normalarray or texturecoordarray.

5.4.4 Vertexpointer

```
vertexpointer size datatype stride arrayname
```

5.4.5 Colorpointer

```
colorpointer size datatype stride arrayname
```

5.4.6 Normalpointer

```
normalpointer size datatype stride arrayname
```

5.4.7 Texcoordpointer

```
texcoordpointer size datatype stride arrayname
```

5.4.8 Arrayelement

```
arrayelement index
```

5.4.9 Drawelements

```
drawelements primitive size arrayname
```

Example

```
begin script
pointsize 10
disable backfacecull

array vertexArray float 9 0 0 0 1 0 0 1 1 1
array vertexArray2 float 12 0 0 0.5 5 1 0 .5 5 1 1 1.5 5
array index int 3 0 1 2

enableclientstate vertexarray
vertexpointer 3 float 0 vertexArray

color 1 0 0
begin triangles
  arrayelement 0
  arrayelement 1
  arrayelement 2
end triangles

array colors float 12 0 0 1 .5 1 0 0 .5 0 1 0 .5

enableclientstate colorarray
```

```
colorpointer 4 float 0 colors
vertexpointer 3 float 4 vertexArray2

enable blend    # enable transparency
depthmask false # should be used before transparency

drawelements triangles 3 index

depthmask true  # should be used after transparency
disable blend


disableclientstate colorarray
disableclientstate vertexarray
end script
```

5.5 Z Buffer Commands

5.5.1 Depthfunc

`depthfunc type`

Sets the comparison function for the depth test. The value for func must be never, always, less, lequal, equal, gequal, greater, or notequal. An incoming fragment passes the depth test if its z value has the specified relation to the value already stored in the depth buffer. The default is less, which means that an incoming fragment passes the test if its z value is less than that already stored in the depth buffer. In this case, the z value represents the distance from the object to the viewpoint, and smaller values mean the corresponding objects are closer to the viewpoint.

5.5.2 Depthmask

`depthmask bool`

Sets the masks used to control writing into the Z buffers.

5.6 Data Transfer Functions

5.6.1 Vertex

```
vertex2f valX valY valZ
v valX valY valZ
v3f valX valY valZ
vertex valX valY valZ
vertex3f valX valY valZ
v4f valX valY valZ valW
vertex4f valX valY valZ valW
```

Send a position in the graphics pipeline.

5.6.2 Color

```
c valR valG valB
c3f valR valG valB
color valR valG valB
color3f valR valG valB
c4f valR valG valB valA
color4f valR valG valB valA
```

Send a color in the graphics pipeline.

5.6.3 Normal

```
n valX valY valZ
n3f valX valY valZ
normal valX valY valZ
normal3f valX valY valZ
```

Send a normal in the graphics pipeline.

5.6.4 Texture coordinate

```
t valU valV
t2f valU valV
texture valU valV
texture2f valU valV
```

Send a texture coordinate in the graphics pipeline.

5.7 Polygon Attributes

5.7.1 Shade Model

```
shademodel mode
```

Sets the shading model. The mode parameter can be either smooth (the default) or flat.

With flat shading, the color of one vertex of a primitive is duplicated across all the primitive's vertices. With smooth shading, the color at each vertex is treated individually. For a line primitive, the colors along the line segment are interpolated between the vertex colors. For a polygon primitive, the colors for the interior of the polygon are interpolated between the vertex colors.

5.7.2 Polygon mode

```
polygonmode face mode
```

Controls the drawing mode for a polygon's front and back faces. The parameter face can be frontandback, front, or back; mode can be point, line, or fill to indicate whether the polygon should be drawn as points, outlined, or filled. By default, both the front and back faces are drawn filled.

For example, you can have the front faces filled and the back faces outlined with two calls to this routine:

```
polygonmode front fill
polygonmode back line
```

5.7.3 Front Face

```
frontface mode
```

Controls how front-facing polygons are determined. By default, mode is ccw, which corresponds to a counterclockwise orientation of the ordered vertices of a projected polygon in window coordinates. If mode is cw, faces with a clockwise orientation are considered front-facing.

5.7.4 Cull Face

```
cullface mode
```

Indicates which polygons should be discarded (culled) before they're converted to screen coordinates. The mode is either front, back, or frontandback to indicate front-facing, back-facing, or all polygons. To take effect, culling must be enabled using enable with cullface; it can be disabled with disable and the same argument.

5.8 States

You can enable/disable several properties of the OpenGL pipeline.

```
enable property
disable property
```

The following properties are available:

```
backfacecull
lighting
texture2d
blend
depthtest
colormaterial
lighting
linestipple
```

The default values are:

```
enable depthtest
enable backfacecull
enable lighting
enable colormaterial
```

5.9 Lines And Points

pointsize, linewidth, linestipple

5.10 Basic 3D Primitives

```
begin primitivename
end primitivename
```

primitive name

```
points
lines
linestrip
lineloop
triangles
trianglefan
trianglestrip
quads
quadstrip
polygon
```

5.11 Transformations

A modeling transformation is used to position and orient the model. For example, you can rotate, translate, or scale the model - or some combination of these operations. To make an object appear further away from the viewer, two options are available - the viewer can move closer to the object or the object can be moved further away from the viewer. Moving the viewer will be discussed later when we talk about viewing transformations. For right now, we will keep the default ”camera” location at the origin, pointing toward the negative z-axis, which goes into the screen perpendicular to the viewing plane.

When transformations are performed, a series of matrix multiplications are actually performed to affect the position, orientation, and scale of the model. You must, however, think of these matrices multiplications occurring in the opposite order from how they appear in the code. The order of transformations is critical. If you perform transformation A and then perform transformation B, you almost always get something different than if you do them in the opposite order. Push and pop matrix

Since the transformations are stored as matrices, a matrix stack provides an ideal mechanism for doing this kind of successive copying, translating, and throwing away. The command pushmatrix copies the matrix on the top of the matrix stack.The matrix contents are, therefore, duplicated in both the top and the second-to-the-top matrices. This top matrix can then be translated and drawn, as desires, and ultimately destroyed using the popmatrix command.

This leaves you right where you were before and ready to repeat the process for the next version of the object. Scaling

5.11.1 Scale

The scaling command scalef multiplies the current matrix by a matrix that stretches, shrinks, or reflects an object along the axes. Each x-, y-, and z-coordinate of every point in the object is multiplied by the corresponding argument x, y, or z. scalef is the only one of the three modeling transformations that changes the apparent size of an object: scaling with values greater than 1.0 stretches an object, and using values less than 1.0 shrinks it. Scaling with a -1.0 value reflects an object across an axis.

```
scalef valX valY valZ
```

5.11.2 Translation

The translation command translatef multiplies the current matrix by a matrix that moves (translates) an object by the given x-, y-, and z-values.

```
translatef valX valY valZ
```

5.11.3 Rotation

The rotation command rotatef multiplies the current matrix that rotates an object in a counterclockwise direction about the ray from the origin through the point (x,y,z). The angle parameter specifies the angle of rotation in degrees. An object that lies farther from the axis of rotation is more dramatically rotated (has a larger orbit) than an object drawn near the axis.

```
rotatef angle valX valY valZ
```

Example

```
pushmatrix

pushmatrix
color3f 1 0 0
translatef 10 10 10
rotatef 45 1 0 0
volume box
popmatrix

pushmatrix
color3f 0 0 1
translatef -5 -5 -5
scalef 2 2 1
volume sphere 2
```

`popmatrix`

`popmatrix`

5.12 Lighting

5.12.1 Enable/Disable lighting

`enable lighting`
`disable lighting`

5.12.2 Enable/Disable light

`enablelight` *lightnumber*
`disablelight` *lightnumber*

5.12.3 Light properties

`light` *properties*

Where properties can be:

- `spotexponent` *lightnumber float*
- `spotcutoff` *lightnumber float*
- `constantattenuation` *lightnumber float*
- `linearattenuation` *lightnumber float*
- `quadraticattenuation` *lightnumber float*
- `ambient` *lightnumber float float float float*
- `diffuse` *lightnumber float float float float*
- `specular` *lightnumber float float float float*
- `position` *lightnumber float float float float*
- `spotdirection` *lightnumber float float float*

5.12.4 Light models

`lightmodel` *properties*

Where properties can be:

- `lightmodellocalviewer` *float*
- `lightmodelambient` *float float float*
- `lightmodeltwoside` *float*

5.13 Color material

5.13.1 Enable/Disable color material

```
enable colormaterial
disable colormaterial
```

5.13.2 Material properties

`material` *face properties*

Where *face* can be:

- `front`
- `back`
- `frontandback`

Where *properties* can be:

- `ambient` *float float float float*
- `diffuse` *float float float float*
- `specular` *float float float float*
- `emission` *float float float float*
- `shininess` *float*
- `ambientanddiffuse` *float float float float*

5.14 Texturing

You need to enable texture mapping before using this feature.

```
bind "filename"
```

Example:

```
enable texture2d
bind "myimage.tif"
begin quads
texture2f 0 0
vertex 0 0 0
texture2f 1 0
vertex 1 0 0
texture2f 1 1
vertex 1 1 0
texture2f 0 1
vertex 0 1 0
end quads
disable texture2d
```

5.15 Multi-Texturing

activetexture, clientactivetexture, multitexcoord2f

5.16 Blending

Blending allows visualizing transparency effects.

5.16.1 Enable/Disable blending

```
enable blend
disable blend
```

Example:

```
enable blend
disable backfacecull
depthmask false

color4f .6 .6 .9 .5
begin quads
vertex3f 0 0 .6
vertex3f 0 1 .6
vertex3f 1 1 .6
vertex3f 1 0 .6
end quads

enable backfacecull
disable blend

depthmak true
```

5.16.2 Blendind function

`blendfunc` *sourcefactor destinationfactor*

The `blendfunc` function specifies pixel arithmetic.

Factor	(f(R),f(G),f(B),f(A))
zero	(0,0,0,0)
one	(1,1,1,1)
srccolor	(R(s)/k(R),G(s)/k(G),B(s)/k(B),A(s)/k(A))
oneinussrccolor	(1,1,1,1)
	(R(s)/k(R),G(s)/k(G),B(s)/k(B),A(s)/k(A))
dstcolor	(R(d)/k(R),G(d)/k(G),B(d)/k(B),A(d)/k(A))
oneminusdstcolor	(1,1,1,1)
srcalpha	(R(d)/k(R),G(d)/k(G),B(d)/k(B),A(d)/k(A))
	(A(s)/k(A),A(s)/k(A),A(s)/k(A),A(s)/k(A))
oneminussrcalpha	(1,1,1,1)
	(A(s)/k(A),A(s)/k(A),A(s)/k(A),A(s)/k(A))
dstalpha	(A(d)/k(A),A(d)/k(A),A(d)/k(A),A(d)/k(A))
oneminusdstalpha	(1,1,1,1)
	(A(d)/k(A),A(d)/k(A),A(d)/k(A),A(d)/k(A))
srcalphasaturate	(i,i,i,1)

5.17 Display list

You must define a display list in the *displaylist* section of your script file.

5.17.1 New list

This function creates a new display list.

`newlist` *listname*

5.17.2 Delete list

This function destroys a display list. It is useful to use this function when you have to manage several display lists.

`deletelist` *listname*

5.17.3 Call list

Calllist allows user to execute a display list in a script (or in an other display list).

`calllist` *listname*

5.17.4 Example

```
#
# display list section
```

```
#
begin displaylist
# create a display list which contains 2 spheres
newlist mylist
pushmatrix
color3f 1 0 0
translatef -2 0 0
volume sphere 2
color3f 0 0 1
translatef 4 0 0
volume sphere 2
popmatrix
endlist              # you must end the display list !!!!
end displaylist

#
# script section
#
begin script

enable backfacecull  # optimization
color3f 0 1 0
volume box

pushmatrix
# translate the coordinate system
translatef 5 5 0
# display the elements
calllist mylist
popmatrix

pushmatrix
# translate and rotate the coordinate system
translatef 5 5 5
rotatef 90 0 1 0
# display the elements
calllist mylist
popmatrix

end script
```

5.18 Clipping planes

A clipping plane specifies a plane against which all geometry is clipped.

5.18.1 Enable/Disable clipping plane

`enableclipplane` *planenumber*
`disableclipplane` *planenumber*

5.18.2 Clipping plane

`clipplane` *planenumber float float float float*

5.19 Specific Functions

5.19.1 Swizzle

The swizzle command allows the components of vectors to be rearranged to form new vectors. The characters x, y, z, and w represent the first, second, third, and fourth components of the original vector, respectively.

This function is useful when we use color space like CIELAB because the representation in this color space is not L*a*b* but a*b*L*.

```
swizzle transformation
```

where transformation can be xyz, yxz, zyx...

Example

```
swizzle zxy
begin points
  vertex 200 10 10
  vertex 200 100 10
  vertex 200 100 100
end points
```

is equal to:

```
begin points
  vertex 10 10 200
  vertex 100 10 200
  vertex 100 100 200
end points
```

5.20 Line stipple

The line stipple specifies the line stipple pattern.

5.20.1 Enable/Disable line stipple

```
enable linestipple
disable linestipple
```

5.20.2 Linestipple function

`linestipple` *factor pattern*

Where *factor* (int) is a multiplier for each bit in the line stipple pattern and *pattern* is a 16-bit integer whose bit pattern determines which fragments of a line will be drawn when the line is rasterized. Bit zero is used first, and the default pattern is all ones.

5.21 Symbolic Constants

5.21.1 Define

The define command allows creating constants (int, float or boolean constants).

```
define name value
```

Example

```
define t1 5
define t2 4
define t3 2

begin points
vertex3f t1 t2 t3
vertex3f t1 t1 t1
vertex3f t3 t2 t1
end points

is equal to:

begin points
vertex3f 5 4 2
vertex3f 5 5 5
vertex3f 2 4 5
end points
```

5.22 Paths

5.22.1 Add Texture Path

This function allows to add a search path in the texture search paths. You can define theses paths using this command or be setting the CDV_TEXTURE_PATH enviroment variable (each paths separate by a ';')

```
addtexturepath "pathname"
```

Example:

```
enable texture2d

addtexturepath "whateveryourwanthere"

bind "myimage.tif"
# if myimage.tif is not in the current directory
# CDV will try to find it the directory named whateveryourwanthere
begin quads
texture2f 0 0
vertex 0 0 0
texture2f 1 0
vertex 1 0 0
texture2f 1 1
vertex 1 1 0
texture2f 0 1
vertex 0 1 0
end quads
disable texture2d
```

6 CDV Scripting Language - Extensions

6.1 Text Visualization

Only one font is available. You must use transformation in order to have your text in a correct orientation and position in the 3D space.

```
text "text to write"
```

6.2 Complex 3D Primitives

6.3 Volumes

```
volume volumetype optional parameters
```

volumetype

```
tetrahedron
box
dodecahedron
icosahedron
cylinder slices top bottom
cone slices bottom
sphere lod (can be 1 2 or 3)
```

Example

```
disable cullface

pushmatrix
color3f 1 0 0
translatef 1 0 0
volume cylinder 20 true true
popmatrix

pushmatrix
color3f 0 1 0
translatef 0 1 0
volume cone 10 false
popmatrix

pushmatrix
color3f 0 0 1
translatef 1 1 0
volume sphere 2
popmatrix
```

7 CDV Scripting Language - Shaders

7.1 Shaders

vertexshader, vertexshaderfromfile, fragmentshader, fragmentshaderfromfile

7.2 Programs

program, useprogram

7.3 Uniform variables

uniform, uniformmatrix