Mini-HOWTO : visualizing 3D data using Octave, VRML and FreeWRL*

Abstract

This document shows how to visualize sets of 3D points, 3D curves, surfaces etc using the numerical language Octave, in conjunction with the VRML browser FreeWRL. The basic functionalities and principles are presented here with examples, while the detailed synopsis can be obtained from the online help system.

Prerequisites: It is assumed that you have installed Octave 2.1.35 (`http://www.octave.org`) or more, FreeWRL (`http://www.crc.ca/FreeWRL`) 0.34 or more (versions between 0.27 and 0.31 may work too) and the Octave-Forge (`http://octave.sourceforge.net`) package. Each one is available from the indicated URL. Rudimentary knowledge of Octave is also assumed.

1 Basics

This document describes some 3D visualization functions from the Octave-Forge package. First, examples are given on how to visualize a surface with the `vmesh()` function and how to visualize a set of 3D points and select a subset with the mouse. The basic usage of FreeWRL, the program that does the 3D visualization are introduced at the same time.

The second section explains more in detail how the VRML package works, presents other functions and shows how to assemble various 3D objects. Finally, a brief summary and plans for the future are given in Section 3.

1.1 Viewing a surface from the Octave prompt

Consider the following Octave commands :

*Author : Etienne Grossmann `<etienne@isr.ist.utl.pt>` (soon replaced by "Octave-Forge developers"?).

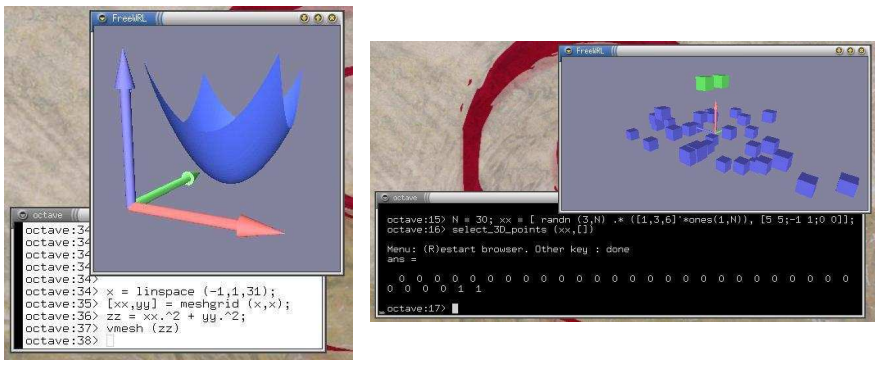

Figure 1: FreeWRL window and octave running in a terminal. The red and green vectors of the frame represent the "X" and "Y" axes. **Left:** Displaying the surface of a 2-variable function. **Right:** Viewing a set of 3D points and selecting a subset.

Listing 1

```
x = linspace (-1,1,31);
[xx,yy] = meshgrid (x,x);
zz = xx.^2 + yy.^2;
vmesh (zz);
```

The first three lines define a 31×31 matrix zz containing values of $x^2 + y^2$ for values of x and y regularly sampled in the interval $[-1, 1]$. The fourth line visualizes in a separate window a surface representing zz[1]. By click and dragging the mouse in that new window, you should be able to rotate the object and obtain something as in Figure 1.

The variant :

```
vmesh (zz,"checker",[5,-2],"col",[1 0 0;0.7 0.7 0.7]');
```

displays (Figure 2, left) a checkered surface with 5 squares along the X direction squares that are two facets wide along the Y direction. This listing can be run with the command `vrml_demo_tutorial_1`. Subsequent in this document listings can be called in a similar fashion.

1.2 Basic 3D visualization with FreeWRL

What happened when this last command was executed is that the program FreeWRL was launched and asked to visualize some data that represents the 3D surface. Interaction with FreeWRL is done through the mouse and keyboard :

[1]If nothing happened when you executed the `vmesh()` command, or if an error message was issued, this indicates a problem. Make sure that you are running under X11 and that FreeWRL is installed and functions properly.

When in "**examine mode**" (the default), dragging with the left button rotates the object, while dragging up and down with the left button zooms out and zooms in.

When the `'w'` key is hit, FreeWRL switches to "**walk mode**" , in which left-dragging up or downwards will move the viewpoint forward or backward. Left-or-right-dragging to the left or right directions will move the viewpoint left or right, respectively. Finally, right-dragging up or down moves the viewpoint up and down. You may switch back to the "examine" mode by hitting the `'e'` key.

Hitting 's' saves a **snapshot** `freewrl.snap.0001.ppm` in the current directory; subsequent snapshots will be numbered 0002, etc. Each time FreeWRL is launched, the count is restarted so beware that previous snapshots may be overwritten.

Last but important, the `'q'` key will **exit** FreeWRL.

1.3 Viewing a set of 3D points

Going back to Octave, let's create a set of 3D points :

Listing 2

```
N = 30;
x = [randn (3,N) .* ([1,3,6]'*ones(1,N)), [5 5;-1 1;0 0]];
select_3D_points (x)
```

The first two commands create a variable `xx` that holds 32 3D points, 30 of which are randomly distributed and two, with coordinates $[5, \pm 1, 0]$, which are "outliers" . The last command launches FreeWRL in such a way that it is not only possible to examine the 3D points, represented by boxes, but also select a subset of them, by clicking on them : a selected box is bright green instead of blue. On the snapshot in Figure 1 (right), the two outliers have been selected.

Having selected these points, let's go back to the Octave window, where `select_3D_points()` has produced the output :

```
Menu: (R)estart browser. Other key : done
```

`select_3D_points()` is now waiting for a key to be hit, either 'r' , in which case FreeWRL will be restarted, or any other key, in which case `select_3D_points()` finishes and returns the indices of the 3D points that have been output, in the form of a 0-1 matrix :

```
ans = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1.
```

2 More in detail

VRML is a language that allows to describe a 3D setup by specifying shapes, colors, positions etc. Given a VRML document, a VRML browser -FreeWRL in our case- will render it and allow a user to navigate in it. Examining 3D objects is just one of the possibilities of VRML, the only one we use in this document.

Communication between Octave and FreeWRL Octave communicates with FreeWRL by writing VRML code to a temporary file

```
/tmp/octave_vrml_output.wrl
```

and sending a signal so that FreeWRL reads the file and raises its window. The Octave function that takes care of this is

```
vrml_browse (str),
```

which takes as argument a single string that contains the VRML code. Another function unfortunately is useful :

```
vrml_kill ().
```

This kills the current browser. Unfortunately, the browser sometimes hangs and it is on that occasion that this function is useful.

Functions for building objects from scratch We now describe some functions that create VRML objects, starting by those used in the first example, Figure 1 (left). The displayed scene consists of four elements :

1. A surface, obtained with the `vrml_surf ()` command, which is the object of interest.
2. A reference frame, obtained with the `vrml_frame ()` command, which indicates the orientation in which the data considered.
3. Some lights , obtained with the `vrml_PointLight ()` command, in order to light the scene.
4. A background, obtained with the `vrml_Background ()` command, to specify the grayish-blue color, rather than the browser's default black.

We now describe each function in turn :

`s = vrml_surf (z,...)` or

`s = vrml_surf (x,y,z,...)` returns a VRML representation of a surface composed of triangular facets whose vertices. The arguments are

`z`	Matrix of dimension $R \times C$ representing the height of the vertices.
`x`	Matrix with dimensions $R \times C$ or vector of length C, containing the abscissa of the vertices. If omitted, the value `linspace (-1,1,C)` is assumed.
`y`	Matrix with dimensions $R \times C$ or vector of length R, containing the ordinate of the vertices. If omitted, the value `linspace (-1,1,R)` is assumed.

extra options can be passed as a key-value pair :

`"tran", tran` Transparency of the surface : `tran` should be a number between 0 and 1.

`"col", col` Color of the surface. The effect of this option depends on the size of `col` :

- 3×1 The RGB color of the surface is uniform and specified by `col`.
- $3 \times RC$ The RGB color of *vertex* (i, j) of the surface is specified by `col(:,i+R*j)` and the color changes smoothly from vertex to vertex.
- $3 \times (R-1)(C-1)$ The RGB color of *face* (i, j) $(1 \leq i \leq R-1,\ 1 \leq j \leq C-1)$ is specified by `col(:,i+(R-1)*j)` and the color change between faces is crisp.

`"checker", sz` Colors the surface as a checker. If `sz` is positive, it is the number of and columns of the checker. If negative, it is the opposite of the number of facets per square of the checker. Moreover, `sz` may have length two, in which case the number of rows and columns are defined separately. If the `"col"` option is used, the first 6 elements (or 2) are the RGB (greylevel) components of the color of the squares. Figure 2, left. Shows the effect of this option.

The options `"col"` and `"checker"` can be passed to the `vmesh` function too. Other options are documented by doing `help vrml_surf`. For building more 3D surfaces consisting of **facets**, see the function `vrml_faces ()`.

`s = vrml_frame (pos, rot, ...)` returns a VRML representation of a XYZ frame.

`pos` 3×1 is the position of the frame. If not provided, $[0\,0\,0]$ is assumed and a `NaN` is ignored.

`rot` 3×1 orientation of the frame. Default is $[0\,0\,0]$ and a `NaN` is ignored.

Translations -or **positions**- will always be represented by a vector of 3 elements. **Orientations** -or **rotations**- are also represented by a 3-element vector whose direction defines the axis of the rotation and whose norm (as returned by `norm(rot)`) defines the angle, **in radians**.

Extra options can be passed as a key-value pair :

`"col", col` $1 \times 3 \text{ or } 3 \times 3$: RGB color of the frame, or of each branch (stacked vertically). Default is $[0.3\,0.4\,0.9]$.

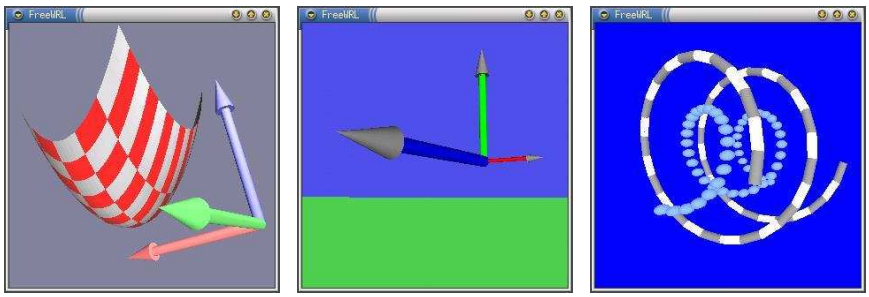

Figure 2: **Left:** A checkered surface obtained by the variant of Listing 1.
Middle: An XYZ frame with axes colored red, blue and green, obtained by running Listing 3.
Right: A set of points connected by black and white cylinders and the same set of points, scaled down by half along the Y and Z axes (Listing 4).

`"hcol", col` 1×3 or 3×3 : RGB color of the heads of the arrows that compose the frame, or of each branch (stacked vertically). Default is $[0.3\,0.4\,0.9]$.

`"scale", scl` 1×1 or 1×3 : Length of the branches of the frame, or of each branch individually. Default is 1.

`s = vrml_PointLight (...)` and

`s = vrml_Background (...)` are low level functions that return simple VRML objects ("nodes", in VRML lingo). All the VRML characteristics ("fields") can be set by using key-value pairs. We describe some of the options that can be passed to `vrml_Background()`; the others are displayed by doing `help vrml_Background()`.

`"skyColor", RGB` 3×1 and

`"groundColor", RGB` 3×1 specify the RGB color of the sky and ground, respectively.

The following listing and Figure 2, middle, show an example of the usage of `vrml_frame()`, `vrml_Background()`, and `vrml_browse()`.

Listing 3

```
s1 = vrml_frame ("scale",[1 2 3],"col",eye (3), "hcol",0.5*[1 1 1]);
s2 = vrml_Background ("skyColor",[3 3 9]/10,"groundColor",[3 8 3]/10);
vrml_browse ([s1,s2]);
```

We now describe the function

`s = vrml_cyl (x,...)` that returns VRML code representing 3D **line segments** (in fact, cylinders) whose extremities are `[x(:,i),x(:,i+1)]`, for `i` in $\{1,..,N\}$ (`x` is a a $3 \times N$ matrix). Amongst other things, the radius, color and transparency can be set. Also, spheres centered at each `x(:,i)` can be added, and an arrow can be used to indicate the last segment. The corresponding options are :

`"rad", rad` The radius of the cylinders linking each pair of consecutive points.

`"col", col` 3×1 or $3 \times N$ The RGB color(s) of the cylinders.

`"tran", col` The transparency of the cylinders.

`"balls"` Add a spheres around each `x(:,i)`.

`"arrow"` Represent the last segment as an arrow.

Positioning, orienting and scaling objects Finally, we present a function that allows to translate, rotate and scale an arbitrary VRML object and is useful for composing 3D setups consisting of many object.

`s = vrml_transfo (str,pos,rot,scale)` translates, rotates and scales the object defined in the string `str`. The arguments `pos` and `rot` represent a translation and rotation respectively, just as in the `vrml_frame()` function. If `scale` is a scalar, the object `str` will be scaled by that amount, while if `scale` is a 3×1 vector, it represents the scaling of the X, Y and Z axes independently.

Listing 4 below and Figure 2 (right) illustrate how the two functions introduced above can be used.

Listing 4

```
x = linspace (0,4*pi,50);
                              # Points on a helix
xx1 = [x/6; sin (x); cos (x)];
                              # Linked by segments
s1 = vrml_cyl (xx1, "col",kron (ones (3,25),[0.7 0.3]));
                              # Scaled and represented by spheres
s2 = vrml_points (xx1,"balls");
s2 = vrml_transfo (s2,nan,[pi/2,0,0],[1 0.5 0.5]);
s3 = vrml_Background ("skyColor",[0 0 1]);
vrml_browse ([s1, s2, s3]);
```

3 Summary and future plans

We have just passed in review the most important aspects of the VRML toolbox. Its main utility is to examine surfaces and moderately sized sets of points, but it can also be used to build and assemble more general 3D objects.

The presented functionality is adapted to my personal needs; if you think of some improvements, please send your suggestions (and perhaps patches) to the mailing list `octave-dev@lists.sourceforge.net`. In the future, I will focus on making more concise VRML code and articulating better the library around that language.