Documentation for Plot3D.h and Plot3D.c

Steven Andrews, © 2003
See the document "LibDoc" for general information about this and other libraries.

```
typedef struct Point3D {
       float r[3];
       float p[3];
       float o[3];
       float s[3];
       float *r2p;
       }* Pt3Dptr;

typedef struct Drawing3D {
       int max;
       int n;
       Pt3Dptr *pt;
       char *col;
       int *line;
       float *r2p;
       }* Draw3Dptr;

int Init3DPlot(int stereo,float dist,float sep);
Pt3Dptr Pt3DAlloc();
void Pt3DFree(Pt3Dptr pt);
void PlotPt3D(Pt3Dptr pt);
void PlotLine3D(Pt3Dptr pt);
void PlotStr3D(Pt3Dptr pt,char *s);
void Rotate3D(char c,float alpha);
void UpdatePt3D(Pt3Dptr pt,int internal);
Draw3Dptr Draw3DAlloc(int n,float *r2p);
void Draw3DFree(Draw3Dptr dr);
int DrawingPt3D(Draw3Dptr dr,float x,float y,float z,int rel,int line,int col);
void PlotDraw3D(Draw3Dptr dr);
void UpdateDraw3D(Draw3Dptr dr,int internal);
float *Draw3DOut(Draw3Dptr dr,char c);
```

Requires: `"Plot.h"`, `"Plot3D.h"`, `"Rn.h"`.

Example Program: `Eulerplot.c`.

Written 2/25/02. The stereo views aren't correct, as seen by rotating the figure 90º down to give a top view. Somehow, the two eye views are correctly rotated, although the eye positions are not dealt with correctly.

This is a relatively easy to use library for drawing simple three dimensional graphics. Pictures can be drawn either as a two dimensional version or as a pair of stereo images. At present, it is useful for observing and manipulating simple objects. However, it is not useful for, say, flying around in a virtual space. The difference is that this library does not support translations or easily defined coordinates of the viewer. There is a tremendous amount that could be done to improve the library, which may be worked on as necessary. Alternatively, OpenGL may be a better solution.

The primary type used here is a 3 dimensional point, pointed to by a `Pt3Dptr`. A point may have a set of relative coordinates (`r`) to describe its position in one reference frame which is rotated relative to the physical reference frame. If this is used, the pointer `r2p` is expected to point to a matrix of direction cosines for expressing the relative rotation. The physical coordinates (`p`) describe where a point actually exists, after the internal rotation was accomplished, if necessary. These are rotated with the global variables `p2o3D` and `p2s3D` to yield the observed (`o`) coordinates and stereo (`s`) coordinates.

A 3 dimensional drawing, pointed to by a `Draw3Dptr`, is a more complex type. It is basically a collection of 3 dimensional points, along with simple instructions that describe whether a pair of points should be connected by a line and what color should be used for the point and/or line. Because the entire drawing is expected to move as a unit, it has an `r2p` pointer, and the default method of use assigns the same rotation matrix to each point in the drawing (note that the drawing does not own the matrix, which should be allocated and freed elsewhere). `max` is the maximum number of points that the drawing may contain, `n` is the number of points actually used, `pt` is the list of points, `col` is the color of the respective point, and `line` is whether a line is drawn to the respective point.

A program should first call `Init3DPlot`, which will call the Plot.c routine `InitPlot` and take care of most setting up required. Then call the Plot.c function `MakeWindow`. After that, you can define points or drawings, update them, plot them, manipulate them, etc. At the end, free the points and drawings, and close the window with `KillWindow`, from Plot.c. It is also possible to try to use events with `InvalPlot` and `EndPlot`.

Global variables

`p2o3D` is the 3x3 rotation matrix from physical coordinates to observed view.
`p2s3D` is the 3x3 rotation matrix from physical coordinates to stereo view.
`work3D1` is a 3x3 rotation matrix for scratch space.
`work3D2` is a 3x3 rotation matrix for scratch space.
`stereo3D` is 1 for a stereo image and 0 if not.
`penpos3Ds` is a 2 element vector of floats to store the pen position in the stereo view, using the Plot.c coordinates, not the absolute pixel coordinates.
`dist3D` is the distance of the viewer from the object.
`sep3D` is the separation of the viewers eyes, used for stereo viewing, both of which are at distance `dist3D`.

Functions

`Init3DPlot` allocates and initiallizes the global variables (with the scope of the whole library, but not beyond it) and initiallizes plotting with the Plot.c function `InitPlot`. Send in 1 for a stereo view and 0 for non-stereo. The return value is usually 0, but might be 1 if memory could not be allocated.
`Pt3DAlloc` allocates a 3 dimensional point and sets all parameters to 0. It returns the point or `NULL` if memory allocation failed.
`Pt3DFree` frees a 3 dimensional point.
`PlotPt3D` plots a point to the screen using the observed and stereo members. That is, the point is not updated for any rotations that may have occured. The Z component of the observed and stereo values is ignored although, of course, it could be used to somehow indicate depth.
`PlotLine3D` draws a line from the current pen position to the point listed. As above, the point is not updated for rotations and the Z component is ignored.
`PlotStr3D` prints the string at the position of the 3 dimensional point.

`Rotate3D` rotates the position of the viewer by angle `alpha`, by modifying the global variables `p2o3D` and `p2s3D`. It does this by multiplying the current rotation matrices with another one that expresses the new rotation. Thus, it is an iterative process and may, with enough rotations, have roundoff errors that yield matrices that are no longer unitary. However, this has not been a problem so far. The characters are the arrow keys (28 is left, 29 is right, 30 is up, and 31 is down) for the obvious rotations and '.' and '/' to rotate counter-clockwise and clockwise in the plane of the screen.

`UpdatePt3D` performs the matrix multiplications to update the coordinates of a point. If `internal` is 0 (or the `r2p` member is `NULL`), the internal rotations are not updated. This is useful to save computation time when a view changes but no relative rotations were done. The observed coordinates and, if necessary, stereo coordinates, are updated.

`Draw3DAlloc` allocates and initiallizes a new 3 dimensional drawing which can contain up to `max` points, as well as the points in it. The drawing `r2p` member and the `r2p` member of each point is set to the `r2p` value that is sent in, which may point to a rotation matrix or be equal to `NULL`. A pointer to the drawing is returned, or `NULL` if memory allocation failed. A new drawing is set to have 0 points defined.

`Draw3DFree` frees a drawing as well as the points in it. As usual, it can also free partially allocated drawings, provided all unallocated portions are equal to `NULL`.

`DrawingPt3D` defines a 3 dimensional point in a drawing. The new point is always the last point of the drawing, and the number of points, `n`, is incremented to account for the new point. If the drawing is already full, in that `n` is equal to `max`, the definition fails and 1 is returned; otherwise 0 is returned. `dr` is a pointer to the drawing; `x`, `y`, and `z` are the point coordinates; `rel` is 1 if these are relative coordinates and 0 if they are physical coordinates; `line` is 1 if a line is to be drawn from the last point to the new point (or the previous pen position, in the case of the first point); and `col` is the color of the new point as well as any line to it.

`PlotDraw3D` plots a 3 dimensional drawing, pointed to by `dr`. As usual, it does not update anything first, and any Z axis information in the `o` and `s` members is ignored. It calls `PlotPt3D` and `PlotLine3D` as needed.

`UpdateDraw3D` updates all the points in a drawing to account for viewer angle changes and any internal rotations. As with `UpdatePt3D`, internal rotations are only updated if `internal` is 1 and the `r2p` matrix is defined for the relevent points. It calls `UpdatePt3D` to take care of the math.

`Draw3DOut` copies the list of data points in the drawing to a matrix, allowing it to be saved easily. The parameter `c` can be '`r`', '`p`', '`o`', or '`s`', to output the relative, physical, observed, or stereo coordinates respectively. The returned result is a matrix allocated to size `dr->n` rows by 3 columns, and filled with the *x*, *y*, and *z* values of the appropriate data point components. `NULL` is returned if allocation failed.