OpenGL and GLUT based library for Image Processing (GLIP)

Pre-Version 0.6
(Perversion)

Antonio Marín-Hernández
Universidad Veracruzana, México &
LAAS – CNRS, France

February 2004

Contents

1 Introduction

The OpenGL and GLUT based library for image processing (GLIP) was developed as a fast and easy way to display singles or sequences of images, as well as to draw in its some primitives and/or sub images. It supports the same functionalities as OpenGL and GLUT. This version is intended to work under all platforms and operating systems that support both OpenGL and GLUT. It could be complemented as well as any other OpenGL or GLUT interface library (i.e. GLUI or GLUIX).

This documentation serves as both as a guide and library specification.

1.1 Background

1.2 Impulse

The main idea behind the construction of this library was the lack of an easier and simple way to display images and some simple primitives, in a device independent system. So following OpenGL philosophy, it intends to make and easier and fast to learn way to display images and results, without worrying about windowing system calls or problems in multiplatform developments.

1.3 Conventions

1.4 Terminology

2 Initialization

2.1 glipInit

`glipinit`

Usage

```
void glipInit(int *argcp, char **argv, int option);
```

`argcp` A pointer to the program's *unmodified* `argc` variable from `main`. Upon return, the value pointed to by `argcp` will be updated, because `glutInit` extracts any command line options intended for the GLUT library.

`argv` The program's unmodified `argv` variable from main. Like `argcp`, the data for `argv` will be updated because `glutInit` extracts any command line options understood by the GLUT library.

`option` Initialization options.

Description

`glipInit` will initialize the GLUT library and negotiate a session with the window system. During this process, `glipInit` may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options.
`glipInit` also processes command line options (as `glutInit`), but the specific options parse are window system dependent.

X Implementation Notes

The X Window System specific options parsed by `glipInit` are as follows (same as for `glutInit`):

`-display` ***DISPLAY*** Specify the X server to connect to. If not specified, the value of the `DISPLAY` environment variable is used.

`-geometry` ***WxH+X+Y*** Determines wherewindow's should be created on the screen. The parameter following `-geometry` should be formatted as a standard X geometry specification. The effect of using this option is to change the GLUT initial size and initial position the same as if `glutInitWindowSize` or `glutInitWindowPosition` were called directly.

`-iconic` Requests all top-level windows be created in an iconic state.

`-indirect` Force the use of indirect OpenGL rendering contexts.

`-direct` Force the use of direct OpenGL rendering contexts (not all GLX implementations support direct rendering contexts). A fatal error is generated if direct rendering is not supported by the OpenGL implementation.

If neither `-indirect` or `-direct` are used to force a particular behavior, GLUT will attempt to use direct rendering if possible and otherwise fallback to indirect rendering.

`-gldebug` After processing callbacks and/or events, check if there are any OpenGL errors by calling `glGetError`. If an error is reported, print out a warning by looking up the error code with `gluErrorString`. Using this option is helpful in detecting OpenGL run-time errors.

`-sync` Enable synchronous X protocol transactions. This option makes it easier to track down potential X protocol errors.

2.2 glutInitDisplayMode

`glutInitDisplayMode` sets the initial display mode (GLUT library function).

Usage

```
void glutInitDisplayMode(unsigned int mode);
```

`mode` Display mode, normally the bitwise *OR*-ing of GLUT display mode bit masks. See values below:

`GLUT_RGBA` Bit mask to select an `RGBA` mode window. This is the default if neither `GLUT_RGBA` nor `GLUT_INDEX` are specified.

`GLUT_RGB` An alias for `GLUT_RGBA`.

`GLUT_INDEX` Bitmask to select a color index mode window. This overrides `GLUT_RGBA` if it is also specified.

`GLUT_SINGLE` Bit mask to select a single buffered window. This is the default if neither `GLUT_DOUBLE` or `GLUT_SINGLE` are specified.

`GLUT_DOUBLE` Bit mask to select a double buffered window. This overrides `GLUT_SINGLE` if it is also specified.

`GLUT_ACCUM` Bit mask to select a window with an accumulation buffer.

`GLUT_ALPHA` Bit mask to select a window with an alpha component to the color buffer(s).

`GLUT_DEPTH` Bit mask to select a window with a depth buffer.

`GLUT_STENCIL` Bit mask to select a window with a stencil buffer.

`GLUT_MULTISAMPLE` Bitmask to select a window with multisampling support. If multisampling is not available, a non-multisampling window will automatically be chosen. Note: both the OpenGL client-side and server-side implementations must support the GLX SAMPLE SGIS extension for multisampling to be available.

`GLUT_STEREO` Bit mask to select a stereo window.

`GLUT_LUMINANCE` Bit mask to select a window with a “luminance” color model. This model provides the functionality of OpenGL’s `RGBA` color model, but the green and blue components are not maintained in the frame buffer. Instead each pixel’s red component is converted to an index between zero and `glutGet(GLUT_WINDOW_COLORMAP SIZE)`-1 and looked up in a per-window color map to determine the color of pixels within the window. The initial colormap of `GLUT_LUMINANCE` windows is initialized to be a linear gray ramp, but can be modified with GLUT’s colormap routines.

Description

The initial display mode is used when creating top-level windows, subwindows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

Note that `GLUT_RGBA` selects the `RGBA` color model, but it does not request any bits of alpha (sometimes called an alpha buffer or destination alpha) be allocated. To request alpha, specify `GLUT_ALPHA`. The same applies to `GLUT_LUMINANCE`.

`GLUT_LUMINANCE` **Implementation Notes**

`GLUT_LUMINANCE` is not supported on most OpenGL platforms.

2.3 glutIdleFunc

2.4 glutMainLoop

`glutMainLoop` enters the GLUT event processing loop.

Usage

```
void glutMainLoop(void);
```

Description

`glutMainLoop` enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

3 Displaying Images

3.1 glipImageSt

`glipImageSt` is the C structure used for manipulate images.

Description

```
typedef struct glipImageSt {

    int width,height;
    int size;
    int depth;
    int NoClrsInPltt;
    enum pixelDataFormat format;
    enum dataType type;
    int signX,signY;
    int byteAlign;
    unsigned char * palette;
    void           * data;
  } imageSt;

enum pixelDataFormat {COLOR_INDEX = 6400,
                RED=6403,
                GREEN,
                BLUE,
                APLHA,
                RGB,
                RGBA,
                LUMINANCE,
                LUMINANCE_ALPHA,
                BGR=32992,
                BGRA};

  enum dataType {BYTE=5120,
            UNSIGNED_BYTE,
            SHORT,
            UNSIGNED_SHORT,
            INT,
            UNSIGNED_INT,
            FLOAT,
```

```
            DOUBLE};
```

3.2 glipCreateImageHeader

```
glipCreateImageHeader
```

Usage

```
imageSt * glipCreateImageHeader(int w,int h,int b, enum
pixelDataFormat format, enum dataType type);
```

Description

3.3 glipCreateImage

```
glipCreateImage
```

Usage

```
imageSt * glipCreateImage(int w,int h,int b,enum pixelDataFormat
format,enum dataType type);
```

Description

3.4 glipDisplayImage

```
glipDisplayImage
```

Usage

```
int glipDisplayImage(imageSt *image, char *name, int winNo)
```

Description

3.5 glipRedisplayImage

```
glipRedisplayImg
```

Usage

```
void glipRedisplayImg(int winNo);
```

Descrption

3.6 glipKillImageWin

glipKillImageWin

Usage

```
int glipKillImageWin(int winNo);
```

Description

4 Displaying Data

4.1 glipDataSt

glipDataSt

Description

```
typedef struct glipDataSt {
    int NoPts;
    int size;
    float *color;   /* checar esto */
    enum lineMode mode;
    enum dataType type;
    int stipple;
    unsigned short pattern;
    void   * pts;
  } glipDataSt;

enum lineMode {POINTS,
            LINES,
            LINE_LOOP,
            LINE_STRIP,
            TRIANGLES,
            TRIANGLE_STRIP,
            TRIANGLE_FAN,
            QUADS,
            QUAD_STRIP,
            POLYGON};
```

4.2 glipCreateDataSt

glipCreateDataSt

Usage

```
glipDataSt * glipCreateDataSt(int NoPts, int dim, int ptSize, float
*color, enum lineMode mode, enum dataType type, void *pts)
```

Description

4.3 glipDeleteData

`glipDeleteData`

Usage

```
Int glipDeleteData()
```

Description

4.4 glipDrawInImage

`glipDrawInImage`

Usage

```
int glipDrawInImage(int win, linesSt *lines)
```

Description

4.5 glipSnapShot

`glipSnapShot`

Usage

```
linesSt
```

Description

5 Changing Defaults Modes

5.1 glutCreateMenu