GLUT Man Pages

glutInit

`glutInit` is used to initialize the GLUT library.

Usage

```
void glutInit(int *argcp, char **argv);
```

`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.

Description

`glutInit` will initialize the GLUT library and negotiate a session with the window system. During this process, `glutInit` 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.

`glutInit` also processes command line options, but the specific options parse are window system dependent.

X Implementation Notes

The X Window System specific options parsed by `glutInit` are as follows:

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

`-geometry` *W* `x` *H* `+` *X* `+` *Y*
: Determines where window'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.

---

glutInitWindowPosition, glutInitWindowSize

`glutInitWindowPosition` and `glutInitWindowSize` set the *initial window position* and *size* respectively.

Usage

```
void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y);
```

`width`
: Width in pixels.

`height`
: Height in pixels.

`x`
: Window X location in pixels.

`y`
: Window Y location in pixels.

Description

Windows created by `glutCreateWindow` will be requested to be created with the current *initial window position* and *size*.

The initial value of the *initial window position* GLUT state is -1 and -1. If either the X or Y component to the *initial window position* is negative, the actual window position is left to the window system to determine. The initial value of the *initial window size* GLUT state is 300 by 300. The *initial window size* components must be greater than zero.

The intent of the *initial window position* and *size* values is to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified size or position. A GLUT program should use the window's reshape callback to determine the true size of the window.

---

glutInitDisplayMode

`glutInitDisplayMode` sets the *initial display mode*.

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`
: Bit mask 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`
: Bit mask 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.

---

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.

---

glutCreateWindow

`glutCreateWindow` creates a top-level window.

Usage

```
int glutCreateWindow(char *name);
```

`name`
: ASCII character string for use as window name.

Description `glutCreateWindow` creates a top-level window. The `name` will be provided to the window system as the window's name. The intent is that the window system will label the window with the name.

Implicitly, the *current window* is set to the newly created window.

Each created window has a unique associated OpenGL context. State changes to a window's associated OpenGL context can be done immediately after the window is created.

The *display state* of a window is initially for the window to be shown. But the window's *display state* is not actually acted upon until `glutMainLoop` is entered. This means until `glutMainLoop` is called, rendering to a created window is ineffective because the window can not yet be displayed.

The value returned is a unique small integer identifier for the window. The range of allocated identifiers starts at one. This window identifier can be used when calling `glutSetWindow`.

X Implementation Notes

The proper X Inter-Client Communication Conventions Manual (ICCCM) top-level properties are established. The `WM_COMMAND` property that lists the command line used to invoke the GLUT program is only established for the first window created.

---

glutPostRedisplay

`glutPostRedisplay` marks the *current window* as needing to be redisplayed.

Usage

```
void glutPostRedisplay(void);
```

Description

Mark the normal plane of *current window* as needing to be redisplayed. The next iteration through `glutMainLoop`, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to `glutPostRedisplay` before the next display callback opportunity generates only a single redisplay callback. `glutPostRedisplay` may be called

within a window's display or overlay display callback to re-mark that window for redisplay.

Logically, normal plane damage notification for a window is treated as a `glutPostRedisplay` on the damaged window. Unlike damage reported by the window system, `glutPostRedisplay` will *not* set to true the normal plane's damaged status (returned by `glutLayerGet(GLUT_NORMAL_DAMAGED)`.

Also, see `glutPostOverlayRedisplay`.

---

glutSwapBuffers

`glutSwapBuffers` swaps the buffers of the *current window* if double buffered.

Usage

```
void glutSwapBuffers(void);
```

Description

Performs a buffer swap on the *layer in use* for the *current window*. Specifically, `glutSwapBuffers` promotes the contents of the back buffer of the *layer in use* of the *current window* to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after `glutSwapBuffers` is called.

An implicit `glFlush` is done by `glutSwapBuffers` before it returns. Subsequent OpenGL commands can be issued immediately after calling `glutSwapBuffers`, but are not executed until the buffer exchange is completed.

If the *layer in use* is not double buffered, `glutSwapBuffers` has no effect.

---

glutCreateMenu

`glutCreateMenu` creates a new pop-up menu.

Usage

```
int glutCreateMenu(void (*func)(int value));
```

`func`

: The callback function for the menu that is called when a menu entry from the menu is selected. The value passed to the callback is determined by the value for the selected menu entry.

Description

`glutCreateMenu` creates a new pop-up menu and returns a unique small integer identifier. The range of allocated identifiers starts at one. The menu identifier range is separate from the window identifier range. Implicitly, the *current menu* is set to the newly created menu. This menu identifier can be used when calling `glutSetMenu`.

When the menu callback is called because a menu entry is selected for the menu, the *current menu* will be implicitly set to the menu with the selected entry before the callback is made.

X Implementation Notes

If available, GLUT for X will take advantage of overlay planes for implementing pop-up menus. The use of overlay planes can eliminate display callbacks when pop-up menus are deactivated. The `SERVER_OVERLAY_VISUALS` convention [5] is used to determine if overlay visuals are available.

---

glutAddMenuEntry

`glutAddMenuEntry` adds a menu entry to the bottom of the *current menu*.

Usage

```
void glutAddMenuEntry(char *name, int value);
```

`name`
: ASCII character string to display in the menu entry.

`value`
: Value to return to the menu's callback function if the menu entry is selected.

Description

`glutAddMenuEntry` adds a menu entry to the bottom of the *current menu*. The string `name` will be displayed for the newly added menu entry. If the menu entry is selected by the user, the menu's callback will be called passing `value` as the callback's parameter.

---

glutAddSubMenu

`glutAddSubMenu` adds a sub-menu trigger to the bottom of the *current menu*.

Usage

```
void glutAddSubMenu(char *name, int menu);
```

`name`
ASCII character string to display in the menu item from which to cascade the sub-menu.

`menu`
Identifier of the menu to cascade from this sub-menu menu item.

Description

`glutAddSubMenu` adds a sub-menu trigger to the bottom of the *current menu*. The string `name` will be displayed for the newly added sub-menu trigger. If the sub-menu trigger is entered, the sub-menu numbered `menu` will be cascaded, allowing sub-menu menu items to be selected.

---

glutDisplayFunc

`glutDisplayFunc` sets the display callback for the *current window*.

Usage

```
void glutDisplayFunc(void (*func)(void));
```

`func`
The new display callback function.

Description

`glutDisplayFunc` sets the display callback for the *current window*. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the *current window* is set to the window needing to be redisplayed and (if no overlay display callback is registered) the *layer in use* is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback (this includes ancillary buffers if your program depends on their state).

GLUT determines when the display callback should be triggered based on the window's redisplay state. The redisplay state for a window can be either set explicitly by calling `glutPostRedisplay` or implicitly as the result of window damage reported by the window system. Multiple posted redisplays for a window are coalesced by GLUT to minimize the number of display callbacks called.

When an overlay is established for a window, but there is no overlay display callback registered, the display callback is used for redisplaying *both* the overlay and normal plane (that is, it will be called if either the redisplay state or overlay redisplay state is set). In this case, the *layer in use* is *not* implicitly changed on entry to the display callback.

See `glutOverlayDisplayFunc` to understand how distinct callbacks for the overlay and normal plane of a window may be established.

When a window is created, no display callback exists for the window. It is the responsibility of the programmer to install a display callback for the window before the window is shown. A display callback *must* be registered for any window that is shown. If a window becomes displayed without a display callback being registered, a fatal error occurs. Passing `NULL` to `glutDisplayFunc` is illegal as of GLUT 3.0; there is no way to ``deregister'' a display callback (though another callback routine can always be registered).

Upon return from the display callback, the *normal damaged* state of the window (returned by calling `glutLayerGet(GLUT_NORMAL_DAMAGED)` is cleared. If there is no overlay display callback registered the *overlay damaged* state of the window (returned by calling `glutLayerGet(GLUT_OVERLAY_DAMAGED)` is also cleared.

---

glutReshapeFunc

`glutReshapeFunc` sets the reshape callback for the *current window*.

Usage

```
void glutReshapeFunc(void (*func)(int width, int height));
```

`func`
: The new reshape callback function.

Description

`glutReshapeFunc` sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The `width` and `height` parameters of the callback specify the new window size in pixels. Before the callback, the *current window* is set to the window that has been reshaped.

If a reshape callback is not registered for a window or `NULL` is passed to `glutReshapeFunc` (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply call `glViewport(0,0,width,height)` on the normal plane (and on the overlay if one exists).

If an overlay is established for the window, a single reshape callback is generated. It is the callback's responsibility to update both the normal plane and overlay for the window (changing the *layer in use* as necessary).

When a top-level window is reshaped, subwindows are not reshaped. It is up to the GLUT program to manage the size and positions of subwindows within a top-level window. Still, reshape callbacks will be triggered for subwindows when their size is changed using `glutReshapeWindow`.

---

glutKeyboardFunc

`glutKeyboardFunc` sets the keyboard callback for the *current window*.

Usage

```
void glutKeyboardFunc(void (*func)(unsigned char key,
                                   int x, int y));
```

`func`
: The new keyboard callback function.

Description

`glutKeyboardFunc` sets the keyboard callback for the *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The `key` callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The `x` and `y` callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing `NULL` to `glutKeyboardFunc` disables the generation of keyboard callbacks.

During a keyboard callback, `glutGetModifiers` may be called to determine the state of modifier keys when the keystroke generating the callback occurred.

Also, see `glutSpecialFunc` for a means to detect non-ASCII key strokes.

---

glutMouseFunc

`glutMouseFunc` sets the mouse callback for the *current window*.

Usage

```
void glutMouseFunc(void (*func)(int button, int state,
                                int x, int y));
```

`func`
: The new mouse callback function.

Description

`glutMouseFunc` sets the mouse callback for the *current window*. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The `button` parameter is one of `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON`, or `GLUT_RIGHT_BUTTON`. For systems with only two mouse buttons, it may not be possible to generate `GLUT_MIDDLE_BUTTON` callback. For systems with a single mouse button, it may be possible to generate only a `GLUT_LEFT_BUTTON` callback. The `state` parameter is either `GLUT_UP` or `GLUT_DOWN` indicating whether the callback was due to a release or press respectively. The `x` and `y` callback parameters indicate the window relative coordinates when the mouse button state changed. If a `GLUT_DOWN` callback for a specific button is triggered, the program can assume a `GLUT_UP` callback for the same button will be generated (assuming the window still has a mouse callback registered) when the mouse button is released even if the mouse has moved outside the window.

If a menu is attached to a button for a window, mouse callbacks will not be generated for that button.

During a mouse callback, `glutGetModifiers` may be called to determine the state of modifier keys when the mouse event generating the callback occurred.

Passing `NULL` to `glutMouseFunc` disables the generation of mouse callbacks.

---

glutIdleFunc

`glutIdleFunc` sets the global idle callback.

Usage

```
void glutIdleFunc(void (*func)(void));
```

Description

`glutIdleFunc` sets the global idle callback to be `func` so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received. If enabled, the idle callback is continuously called when events are not being received. The callback routine has no parameters. The *current window* and *current menu* will not be changed before the idle callback. Programs with multiple windows and/or menus should explicitly set the *current window* and/or *current menu* and not rely on its current setting.

The amount of computation and rendering done in an idle callback should be minimized to avoid affecting the program's interactive response. In general, not more than a single frame of rendering should be done in an idle callback.

Passing `NULL` to `glutIdleFunc` disables the generation of the idle callback.

---

glutTimerFunc

`glutTimerFunc` registers a timer callback to be triggered in a specified number of milliseconds.

Usage

```
void glutTimerFunc(unsigned int msecs,
                   void (*func)(int value), value);
```

Description

`glutTimerFunc` registers the timer callback `func` to be triggered in at least `msecs` milliseconds. The `value` parameter to the timer callback will be the value of the `value` parameter to `glutTimerFunc`. Multiple timer callbacks at same or differing times may be registered simultaneously.

The number of milliseconds is a lower bound on the time before the callback is generated. GLUT attempts to deliver the timer callback as soon as possible after the expiration of the callback's time interval.

There is no support for canceling a registered callback. Instead, ignore a callback based on its `value` parameter when it is triggered.

---

glutSolidSphere, glutWireSphere

`glutSolidSphere` and `glutWireSphere` render a solid or wireframe sphere respectively.

Usage

```
void glutSolidSphere(GLdouble radius,
                     GLint slices, GLint stacks);
void glutWireSphere(GLdouble radius,
                    GLint slices, GLint stacks);
```

`radius`
: The radius of the sphere.

`slices`
: The number of subdivisions around the Z axis (similar to lines of longitude).

`stacks`
: The number of subdivisions along the Z axis (similar to lines of latitude).

Description

Renders a sphere centered at the modeling coordinates origin of the specified `radius`. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

---

glutSolidCube, glutWireCube

`glutSolidCube` and `glutWireCube` render a solid or wireframe cube respectively.

Usage

```
void glutSolidCube(GLdouble size);
void glutWireCube(GLdouble size);
```

Description

`glutSolidCube` and `glutWireCube` render a solid or wireframe cube respectively. The cube is centered at the modeling coordinates origin with sides of length `size`.

---

glutSolidCone, glutWireCone

`glutSolidCone` and `glutWireCone` render a solid or wireframe cone respectively.

Usage

```
void glutSolidCone(GLdouble base, GLdouble height,
                   GLint slices, GLint stacks);
void glutWireCone(GLdouble base, GLdouble height,
                  GLint slices, GLint stacks);
```

`base`
: The radius of the base of the cone.

`height`
: The height of the cone.

`slices`
: The number of subdivisions around the Z axis.

`stacks`
: The number of subdivisions along the Z axis.

Description

`glutSolidCone` and `glutWireCone` render a solid or wireframe cone respectively oriented along the Z axis. The base of the cone is placed at Z = 0, and the top at Z = `height`. The cone is subdivided around the Z axis into slices, and along the Z axis into stacks.

---

glutSolidTorus, glutWireTorus

`glutSolidTorus` and `glutWireTorus` render a solid or wireframe torus (doughnut) respectively.

Usage

```
void glutSolidTorus(GLdouble innerRadius,
                    GLdouble outerRadius,
                    GLint nsides, GLint rings);
void glutWireTorus(GLdouble innerRadius,
                   GLdouble outerRadius,
                   GLint nsides, GLint rings);
```

`innerRadius`
: Inner radius of the torus.

`outerRadius`

Outer radius of the torus.

`nsides`
Number of sides for each radial section.

`rings`
Number of radial divisions for the torus.

Description

`glutSolidTorus` and `glutWireTorus` render a solid or wireframe torus (doughnut) respectively centered at the modeling coordinates origin whose axis is aligned with the Z axis.

---

glutSolidDodecahedron, glutWireDodecahedron

`glutSolidDodecahedron` and `glutWireDodecahedron` render a solid or wireframe dodecahedron (12-sided regular solid) respectively.

Usage

```
void glutSolidDodecahedron(void);
void glutWireDodecahedron(void);
```

Description

`glutSolidDodecahedron` and `glutWireDodecahedron` render a solid or wireframe dodecahedron respectively centered at the modeling coordinates origin with a given radius.

---

glutSolidOctahedron, glutWireOctahedron

`glutSolidOctahedron` and `glutWireOctahedron` render a solid or wireframe octahedron (8-sided regular solid) respectively.

Usage

```
void glutSolidOctahedron(void);
void glutWireOctahedron(void);
```

Description

`glutSolidOctahedron` and `glutWireOctahedron` render a solid or wireframe octahedron respectively centered at the modeling coordinates origin with a radius of 1.0.

---

glutSolidTetrahedron, glutWireTetrahedron

`glutSolidTetrahedron` and `glutWireTetrahedron` render a solid or wireframe tetrahedron (4-sided regular solid) respectively.

Usage

```
void glutSolidTetrahedron(void);
void glutWireTetrahedron(void);
```

Description

`glutSolidTetrahedron` and `glutWireTetrahedron` render a solid or wireframe tetrahedron respectively centered at the modeling coordinates origin with a given radius.

---

glutSolidIcosahedron, glutWireIcosahedron

`glutSolidIcosahedron` and `glutWireIcosahedron` render a solid or wireframe icosahedron (20-sided regular solid) respectively.

Usage

```
void glutSolidIcosahedron(void);
void glutWireIcosahedron(void);
```

Description

`glutSolidIcosahedron` and `glutWireIcosahedron` render a solid or wireframe icosahedron respectively. The icosahedron is centered at the modeling coordinates origin and has a radius of 1.0.

---

glutSolidTeapot, glutWireTeapot

`glutSolidTeapot` and `glutWireTeapot` render a solid or wireframe teapot respectively.

Usage

```
void glutSolidTeapot(GLdouble size);
void glutWireTeapot(GLdouble size);
```

`size`
: Relative size of the teapot.

Description

`glutSolidTeapot` and `glutWireTeapot` render a solid or wireframe teapot respectively. Both surface normals and texture coordinates for the teapot are generated. The teapot is generated with OpenGL evaluators.