



Lecture overview

- 1) GLUT philosophy and basis
- 2) Initialization
- 3) Event process
- 4) Window management
- 5) Game mode
- 6) State retrieval
- 7) Fonts
- 8) Menu management
- 9) Geometric shapes
- 10) Hints and suggestions



The OpenGL Utility Toolkit (GLUT) is an API for writing window system independent OpenGL programs.

GLUT supports:

- Multiple **windows** for OpenGL rendering.
- Callback driven **event processing**.
- An “**idle**” routine and **timers**.
- A simple, cascading **pop-up menus**.
- Routines to generate solid and wire frame **objects**.
- **Overlay** support
- Support for bitmap and stroke **fonts**.



Program design

typical program structure

- 1) Initialize OpenGL color mode
- 2) Set windows' properties
- 3) Create menus
- 4) Set callbacks
- 5) Enter the main loop of the windowing system



What are the callbacks?

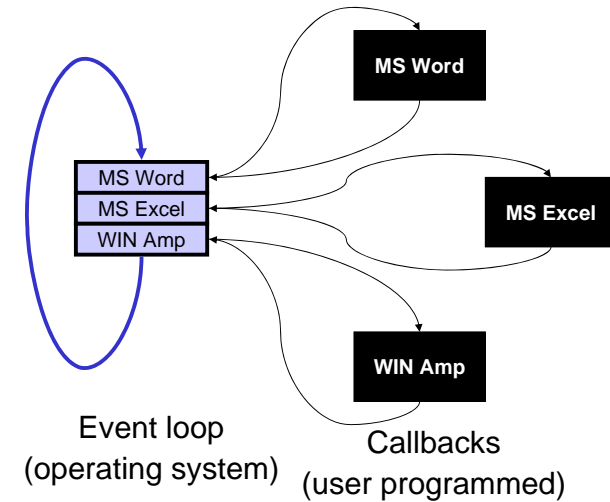
Windowing systems are based on *events*
an event could be:

- opening a window
- mouse move
- hit a key
- closing a window
- selecting an item from a menu
- etc.

A callback is a piece of code that serves for corresponding event



What are the callbacks?



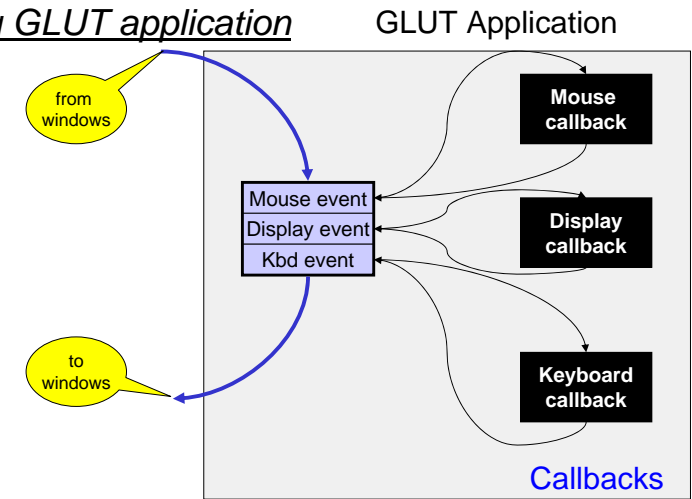
(C) Bedrich Benes

5



What are the callbacks?

Creating GLUT application



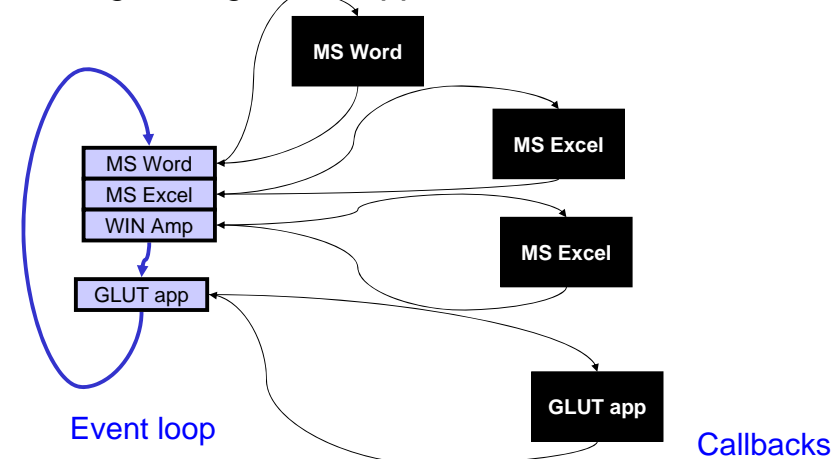
(C) Bedrich Benes

6



What are the callbacks?

Registering GLUT application



(C) Bedrich Benes

7



What are the callbacks?

The "best" way how to hang an application

```
void callback(void){
    while(1);
}
```

The best way how to exit an application

```
void KbdCallback(unsigned char key,
                 GLint x, GLint y){
    if (key==27) exit(0);
}
```

(C) Bedrich Benes

8



Program design

typical structure of main

```
int main(int argc, char *argv[]){
    glutInitDisplayMode(mode); //set properties
    glutInit(argc,argv); //negotiate the window system
    SetWindowProperties(); //size and position
    glutCreateWindow("My App"); //display Window
    CreateMenus(); //create cascading menus
    SetCallbacks(); //set routines
    glutMainLoop(); //register the application
    return 0; //unreachable code (why?)
}
```



2. Initialization

```
void glutInit(int *argc, char *argv[]);
```

where

argc program's argc variable from main
argv program's argv variable from main

- initializes the OpenGL application, GLUT, and negotiates a session with the window system.
- may terminate the program with an error message e.g., when system does not support OpenGL
- processes command line options, the options are window system dependent.



2. Initialization (contd.)

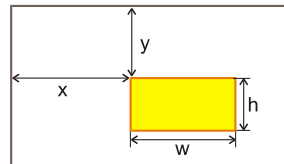
```
void glutInitWindowSize(int w,int h)
void glutInitWindowPosition(int x, int y);
```

where

w is width of the window in pixels
h is height of the window in pixels
x, y is window location in pixels

- if x, y are negative the initial position is left to windows
- default values are:

x = y = -1
w = h = 300



2. Initialization (contd.)

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

m specifies display mode bitwise OR-ing of GLUT display mode bit masks.

GLUT_RGBA	RGBA mode window (default)
GLUT_INDEX	INDEX mode (overrides RGBA)
GLUT_RGB	an alias for GLUT_RGBA.
GLUT_SINGLE	single buffered window (default)
GLUT_DOUBLE	double buffered window
GLUT_ACCUM	accumulation buffer
GLUT_ALPHA	alpha component
GLUT_DEPTH	depth buffer
GLUT_STENCIL	stencil buffer



2. Initialization (contd.)

GLUT_STEREO	stereo displaying
GLUT_MULTISAMPLE	multisampling support (antialiasing). Must be supported by hw (GLX_SAMPLE_SGIS extension).
GLUT_LUMINANCE	red component is converted to an index between 0 and <code>glutGet(GLUT_WINDOW_COLORMAP_SIZE)-1</code> initially - linear colormap, can be modified



2. Initialization (contd.)

Note:

GLUT_RGBA selects the color mode but does NOT allocate any bits for alpha. This must be specified by GLUT_ALPHA

Example:

```
glutInitDisplayMode(GLUT_RGBA | GLUT_ALPHA |
                   GLUT_DEPTH |
                   GLUT_SINGLE);
```

specifies the most frequently used mode i.e., RGBA color mode with alpha for transparency, single buffered, and Z-buffer.



2. Initialization (contd.)

Single vs. double buffer

- single buffer is usual way of rendering static scenes
- double buffer (also called *off-screen rendering*)
 - ~ image is rendered into memory and then it is swapped
 - ~ useful for dynamical scene displaying

→ we do not see the scene while it is rendered!

```
void glutSwapBuffers(void)
```

swaps the memory and the frame buffer

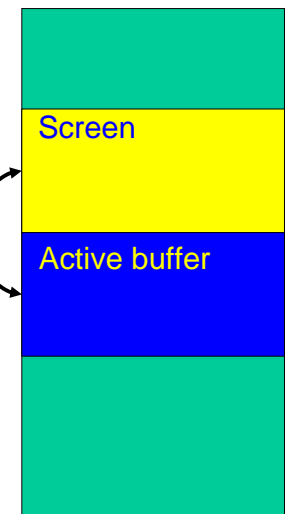


2. Initialization (contd.)

Single vs. double buffer (contd.)

typical animation example:

```
while (!EndOfAnimation()) {
    RenderScene();
    glutSwapBuffers();
}
```



Video memory

2. GLUT initialization



```
void glutInitDisplayString(char *mode)
```

describes the display mode via a string

alpha>=1	sets alpha>=1 bit per pixel
acca	accumulation buffer and alpha
acc	accumulation buffer
red, green, blue	color buffers
buffer	# of bits for indexed color mode
depth	Z-buffer (default>=12)
double	yes or not (default=1)
rgba, rgb	# of pixels for the mode
luminance	# of bits in red (rgb luminance mode)
stencil	# of bits in stencil
stereo	yes or not (default=1)

2. GLUT initialization



samples
win32pfd

multisampling mode (default<=4)
only win32, matches the
Pixel Format Description by number

Example:

"stencil~2 rgb double depth>=16 samples"
RGBA color model (but no bits of alpha), a depth buffer
with at least 16 bits of precision but preferring more,
multisampling if available, and at least 2 bits of stencil
(favoring less stencil to more as long as 2 bits are
available).

"stencil>=2 rgb double depth samples"
stencil with two or more bits, rgba with no alpha, depth
buffer and multisampling

3. Event process



Registering an application

```
void glutMainLoop()
```

- enters the event processing loop
- should be called at most once
- this routine will never return
- causes OS to call callbacks that have been registered

3. Event process



Callback registration - display and windows

```
void glutDisplayFunc(void (*foo)(void))
```

function void foo(void) is called when the window is
(re)displayed (minimized, etc.)

better specified:

if glutGet (GLUT_NORMAL_DAMAGED) returns 1, there is a
need for redisplay and the routine will be called
It can be also specified by user calling

```
void glutPostRedisplay(void)
```



Callback registration - display and windows

```
void glutIdleFunc(void (*foo)(void))
```

callback is called when no event is generated



Fast and effective displaying? Approach 1)

```
void Display(void){
    int i;
    for (i=0;i<360;i++){
        glRotate(1,0,1,1);
    }
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glutWireTeapot(1.0);
    glutSwapBuffer();
}

void main(void){
    ...
    glutDisplayFunc(Display);
    ...
}
```

~ uses 100% CPU
~ speed of displaying depends on CPU
~ application does NOT respond!



Fast and effective displaying? Approach 2)

```
void Idle(void){
    glRotate(1,0,1,1);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glutWireTeapot(1.0);
    glutSwapBuffer();
}

void main(void){
    ...
    glutIdleFunc(Idle);
    ...
}
```

~ uses 100% CPU
~ speed of displaying depends on CPU
~ application responds
~ idle simulates infinite cycle



Fast and effective displaying? Approach 3)

```
void Display(void){
    glRotate(1,0,1,1);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glutWireTeapot(1.0);
    glutSwapBuffer();
}

void Idle(void){
    if (NeedDisplay()) glutPostRedisplay();
}

void main(void){
    ...
    glutIdleFunc(Idle);
    glutDisplayFunc(Display);
    ...
}
```

~ uses minimum CPU
~ speed of displaying depends on CPU
~ application responds



CPU speed independent application?

- 1) Read the system time
- 2) Calculate the position of the objects
- 3) Display the scene

routines for reading the time

```
#include <time.h>
```

```
clock_t t;
```

```
//t is time in milliseconds
```

```
//since the first call within the program
```

```
//typically clock_t is long int
```

Constant speed v the distance s depends on the time

$$v = s/t$$

so the position in the space is

$$s = v * t$$

glutGet(GLUT_ELAPSED_TIME) - similar



Callback registration - display and windows

```
void glutReshapeFunc(void (*foo)(int w,int h))
```

function void foo(int w, int h) is called when the size of the window is changed w and h are the new window size

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

function void foo(unsigned char key,int x,int y) is called when a key is pressed. key is the pressed key and x and y indicate the mouse position
Works only for ASCII codes!



Callback registration - display and windows

```
void glutVisibilityFunc(void (*foo)(int state))
```

function void foo(int w, int h) is called when the visibility of a window has changed

state is GLUT_NOT_VISIBLE if NO pixel is visible and GLUT_VISIBLE otherwise



Callback registration - keyboard

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

function void foo(unsigned char key,int x,int y) is called when a key is pressed. key is the pressed key and x and y indicate the mouse position
Works only for ASCII codes!



This is NOT a callback!

```
int glutGetModifiers(void)
```

returns

GLUT_ACTIVE_SHIFT or

GLUT_ACTIVE_CTRL or

GLUT_ACTIVE_ALT

this should be maintained INSIDE callback,
that is associated to keyboard events



Callback registration - keyboard

```
void glutSpecialFunc(void (*foo)(int key,  
                                int x,int y))
```

x and y indicate the mouse position

key is

GLUT_KEY_F1,..., GLUT_KEY_F12,

GLUT_KEY_LEFT,_RIGHT,

GLUT_KEY_UP,_DOWN

GLUT_KEY_PAGE_UP,..., _DOWN,

GLUT_KEY_HOME,_END,_INSERT

note:

ESC, backspace, and delete are ASCII characters



Callback registration - mouse

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

function void foo(int button,int state,
int x,int y)

is called when button is pressed.

button indicates the button

GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON

state indicates whether is pressed or released

GLUT_UP, GLUT_DOWN

x and y indicate the mouse position



Callback registration - mouse

```
void glutEntryFunc(void (*foo)(int state))
```

callback is called when the focus of the window is changed

GLUT_ENTERED or GLUT_LEFT the window



Callback registration - mouse

```
void glutMotionFunc(void (*foo)(int x, int y))
void glutPassiveMotionFunc( -"- )
```

callback is called when the mouse is moving with pressed button in case of `glutMotionFunc`
or just simply moving in case of `glutPassiveMotionFunc`
`x` and `y` indicates mouse coordinates

useful namely for walkthrough of a scene...



Example: an object rotates and the direction of rotation is changed when the mouse is pressed

```
GLfloat angle = 3; //global variable...
void Mouse(int button, int state, int x, int y){
    if (state == GLUT_DOWN) angle = -angle;
}
```

```
void Idle(void){
    glRotatef(angle,0,0,1);
    RenderScene();
    glutSwapBuffers();
}
```

```
int main(int argc, char **argv){
    glutIdleFunc(Idle);
    glutMouseFunc(Mouse);
}
```



Callback registration - timers

```
void glutTimerFunc(unsigned int msec,
                   void (*foo)(int val), val))
```

`msec` # of milliseconds to pass before the call of `foo`
`foo` the callback itself
`value` an integer value to pass inside the callback

- ~ after registering the callback will be called after `msec` milliseconds
- ~ more callbacks for the same time can be registered
- ~ no way to erase (but you can decide with the value)
- ~ another way of cycle generation - plan timer inside the callback of timer



Windows and sub-windows

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

- creates *top level* window called `name`
- returns unique number
- rendering before entering `glutMainLoop` has no effect

```
int glutCreateSubWindow(int win,
                       int x,int y,
                       int w, int h)
```

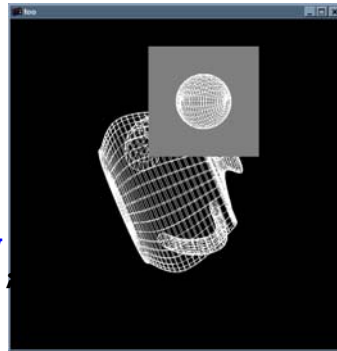
creates sub window under `win` at relative position `[x,y]`
`w` `x` `h` pixels large



Example: sub-window

```
//main window size
glutInitWindowSize(600, 600);
//remember number of the window
int x = glutCreateWindow("foo");
//set callback to main window
glutDisplayFunc(display1);
//make sub window
glutCreateSubWindow(x,250,50,200,200);
//set callback to sub window
glutDisplayFunc(display2);
//run it
glutMainLoop();
```

(C) Bedrich Benes



37



Example: two main windows

```
glutInitWindowSize(400,400);//first window size and
glutInitWindowPosition(100,100);//position
glutCreateWindow("Window 1");//make it and
glutReshapeFunc(Reshape);
glutDisplayFunc(Display1);//set display etc.
glutMouseFunc(Mouse);
glutIdleFunc(Idle1);
```

```
glutInitWindowSize(400,400);
glutInitWindowPosition(500,100);
glutCreateWindow("Window 2");
glutReshapeFunc(Reshape);
glutDisplayFunc(Display2);
glutKeyboardFunc(Kbd);
glutMouseFunc(Mouse);
glutIdleFunc(Idle2);
glutMainLoop();
```



(C) Bedrich Benes

38



```
void glutSetWindow(int n)
```

sets the window *n* as an active window

```
int glutGetWindow(void)
```

gets the number of the active Window

```
void glutDestroyWindow(int n)
```

destroys the window *n*

(C) Bedrich Benes

39



```
void glutPositionWindow(int x,int y)
```

changes position of the current window

```
void glutReshapeWindow(int w,int h)
```

changes size of the window

```
void glutFullScreen(void)
```

very useful!

(C) Bedrich Benes

40



```
void glutPopWindow(void)
void glutPushWindow(void)
void glutShowWindow(void)
void glutHideWindow(void)
void glutIconifyWindow(void)
```

clear from the names...

```
glutSetWindowTitle(char *name)
glutSetIconTitle(char *name)
```

sets (changes) names of the window and icon



```
void glutSetCursor(int cursor)
```

sets the cursor to one of:

```
GLUT_CURSOR_RIGHT_ARROW, GLUT_CURSOR_LEFT_ARROW,
GLUT_CURSOR_INFO, GLUT_CURSOR_DESTROY, GLUT_CURSOR_HELP,
GLUT_CURSOR_CYCLE, GLUT_CURSOR_SPRAY, GLUT_CURSOR_WAIT
GLUT_CURSOR_TEXT, GLUT_CURSOR_CROSSHAIR,
GLUT_CURSOR_UP_DOWN, GLUT_CURSOR_LEFT_RIGHT,
GLUT_CURSOR_TOP_SIDE, GLUT_CURSOR_BOTTOM_SIDE,
GLUT_CURSOR_LEFT_SIDE, GLUT_CURSOR_RIGHT_SIDE,
GLUT_CURSOR_TOP_LEFT_CORNER, GLUT_CURSOR_TOP_RIGHT_CORNER,
GLUT_CURSOR_BOTTOM_RIGHT_CORNER,
GLUT_CURSOR_BOTTOM_LEFT_CORNER, GLUT_CURSOR_INHERIT,
GLUT_CURSOR_NONE, GLUT_CURSOR_FULL_CROSSHAIR
```

“Game” mode:

```
glutSetCursor(GLUT_CURSOR_NONE);
glutFullScreen();
```



Since GLUT 3.7 the game mode is supported
 ~ extremely fast, using many simplifications
 ~ support for joystick
 ~ better keyboard access

```
int glutGameModeGet(Glenum info)
```

```
GLUT_GAME_MODE_ACTIVE      ~ 0 or 1
GLUT_GAME_MODE_POSSIBLE   ~ 0 or 1
GLUT_GAME_MODE_WIDTH      ~ screen
GLUT_GAME_MODE_HEIGHT
GLUT_GAME_MODE_PIXEL_DEPTH ~ # of bits per pixel
GLUT_GAME_MODE_REFRESH_RATE ~ in Hz
GLUT_GAME_MODE_DISPLAY_CHANGED ~ 0 or 1
```



```
int glutEnterGameMode(void)
int glutLeaveGameMode(void)
```

~ enters (leaves) high-performance fullscreen rendering
 ~ no menus
 ~ just one window
 ~ can be called again to change the mode
 ~ it is NOT glutFullScreen()



```
void glutGameModeString(char *mode)
```

sets the game mode configuration

Examples:

"640x480:16@60"

"800x600:16@60"



```
void glutForceJoystickFunc(void)
```

calls the joystick callback (good in the Idle)

```
void glutJoystickFunc(void (*foo)(unsigned
int buttonMask,int x, int y, int z), int
pollInterval);
```

sets the joystick callback:

pollInterval	how frequently it is scanned [msec]
buttonMask	GLUT_JOYSTICK_BUTTON_A,..._D
x,y,z	axes of the joystick (-1000...1000)



```
glutIgnoreKeyRepeat(int ignore)
```

non-zero - disables the autorepeat

zero - enables

typically used with

glutKeyboardUpFunc and glutSpecialUpFunc

glutDeviceGet(GLUT_DEVICE_IGNORE_KEY_REPEAT)
gets the state



```
glutKeyboardUpFunc(void (*foo)(unsigned
char key,int x, int y))
glutSpecialUpFunc(void (*foo)(unsigned
char key,int x, int y))
```

similar to the glutKeyboardFunc but is generated when the key is released

```
void glutSetKeyRepeat(int mode)
```

mode: GLUT_KEY_REPEAT_OFF , _ON, _DEFAULT



```
int glutGet(GLenum state)
```

returns a state of GLUT

GLUT_WINDOW_X X location of the *current window*.
 GLUT_WINDOW_Y Y location of the *current window*.
 GLUT_WINDOW_WIDTH Width in pixels of the *current window*.
 GLUT_WINDOW_HEIGHT Height in pixels of the *current window*.
 GLUT_WINDOW_BUFFER_SIZE # of bits for *current window's* color buffer.
 GLUT_WINDOW_STENCIL_SIZE # of bits in the *current window's* stencil buffer.
 GLUT_WINDOW_DEPTH_SIZE # of bits in the *current window's* depth buffer.
 GLUT_WINDOW_RED_SIZE # of bits of red stored the *current window's* color
 (similarly for green, blue, and alpha)
 GLUT_WINDOW_ACCUM_RED_SIZE # of bits of red stored in the accumulation buffer
 GLUT_WINDOW_DOUBLEBUFFER one if the *current window* is double buffered
 GLUT_WINDOW_RGBA one if the *current window* is RGBA mode
 GLUT_WINDOW_PARENT The windownumber of the *current window's* parent
 zero if the window is a top-level window.
 GLUT_WINDOW_NUM_CHILDREN The number of non-recursive subwindows
 GLUT_WINDOW_NUM_SAMPLES Number of samples for multisampling
 GLUT_WINDOW_STEREO One if the *current window* is stereo, zero otherwise.
 GLUT_WINDOW_CURSOR Current cursor for the *current window*.
 GLUT_SCREEN_WIDTH Width of the screen in pixels
 GLUT_SCREEN_HEIGHT Height of the screen in pixels. Zero the height is unknown



GLUT_SCREEN_WIDTH_MM Width of the screen in millimeters.
 GLUT_SCREEN_HEIGHT_MM Height of the screen in millimeters.
 Zero indicates the height is unknown or not available.
 GLUT_MENU_NUM_ITEMS Number of menu items in the *current menu*.
 GLUT_DISPLAY_MODE_POSSIBLE the *current display mode* is supported or not.
 GLUT_INIT_DISPLAY_MODE The *initial display mode* bit mask.
 GLUT_INIT_WINDOW_X The X value of the *initial window position*.
 GLUT_INIT_WINDOW_Y The Y value of the *initial window position*.
 GLUT_INIT_WINDOW_WIDTH The width value of the *initial window size*.
 GLUT_INIT_WINDOW_HEIGHT The height value of the *initial window size*.
 GLUT_ELAPSED_TIME Number of milliseconds since glutInit called
 (or first call to glutGet(GLUT_ELAPSED_TIME))



```
int glutDeviceGet(GLenum device)
```

detects if a device is present and its properties

GLUT_HAS_KEYBOARD	0 or 1
GLUT_HAS_MOUSE	0 or 1
GLUT_HAS_SPACEBALL	0 or 1
GLUT_HAS_DIAL_AND_BUTTON_BOX	0 or 1
GLUT_HAS_TABLET	0 or 1
GLUT_NUM_MOUSE_BUTTONS	number
GLUT_NUM_SPACEBALL_BUTTONS	number
GLUT_NUM_DIALS	number
GLUT_NUM_TABLET_BUTTONS	number



```
int glutGetModifiers(void)
```

returns modifiers that is activated

GLUT_ACTIVATE_SHIFT,
 GLUT_ACTIVATE_CTRL
 GLUT_ACTIVATE_ALT

~ usually used in the keyboard callback
 ~ can be bitwise and-ed e.g.,

GLUT_ACTIVATE_SHIFT & GLUT_ACTIVATE_CTRL



```
int glutExtensionSupported(char *extension)
```

checks if the `*extension` is supported

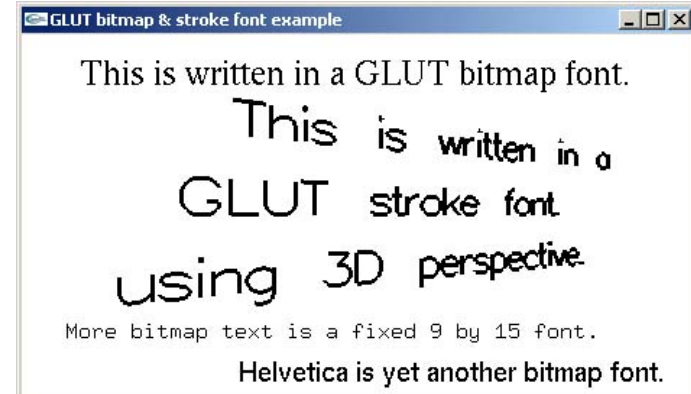
another way to get list of all extensions supported is

```
char *glGetString(GL_EXTENSIONS)
```



GLUT supports two kinds of fonts

- 1) Bitmaps - of bits, bad to zoom, rotate, etc.
- 2) Stroke fonts - from line segments, nicer, but more CPU demanding



```
glutBitmapCharacter(void *font,int character)
```

font is one of the:

```
GLUT_BITMAP_8_BY_13
GLUT_BITMAP_9_BY_15
GLUT_BITMAP_TIMES_ROMAN_10
GLUT_BITMAP_TIMES_ROMAN_24
GLUT_BITMAP_HELVETICA_10
GLUT_BITMAP_HELVETICA_12
GLUT_BITMAP_HELVETICA_18
```

```
int glutBitmapWidth(GLUTbitmapfont f,int c)
```

returns width of a character `c` of the font `f` in pixels



```
glutStrokeCharacter(void *font,int character)
```

font is one of the:

```
GLUT_STROKE_ROMAN
GLUT_STROKE_ROMAN_MONO
```

```
int glutStrokeWidth(GLUTstrokefont f,int c)
```

returns width of a character `c` of the font `f` in pixels



6. Menu management

- GLUT supports simple cascading pop-up menus
- functionality is simple and minimalistic
- not for a full-featured user interface
- it is illegal to create, destroy, change, add, or remove menu items while menus are in use



6. Menu management (contd.)

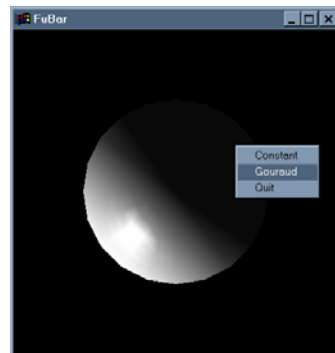
Example:

```
void MakeMenu(){
    //create menu and assign callback
    glutCreateMenu(Menu);
    //attach an action to menu
    glutAttachMenu(GLUT_LEFT_BUTTON);
    //define actions
    glutAddMenuEntry("Constant", 1);
    glutAddMenuEntry("Gouraud", 2);
    glutAddMenuEntry("Quit", 3);
}
```



6. Menu management (contd.)

```
void Menu(int val){
    switch (val){
        case 1: glShadeMode(GL_FLAT);break;
        case 2: glShadeMode(GL_SMOOTH);break;
        case 3: exit(0);break;
    }
    glutSolidSphere();
}
```



6. Menu management (contd.)

```
int glutCreateMenu(void (*foo)(int val))
```

```
void foo(int val)
```

- creates new menu
- foo is called when a menu entry is selected
- val is passed to foo when menu is selected
- returns unique value - menu identifier



6. Menu management (contd.)

```
void glutSetMenu(int menu)
int  glutGetMenu(void)
```

- sets/gets the current menu

```
void glutDestroyMenu(int menu)
```

- destroys the menu
- if the menu is the current one
returns zero and performs no task



6. Menu management (contd.)

```
void glutAddMenuEntry(char *name, int val)
```

name ASCII character string to display in the menu entry.
val value to return to the menu's callback function if the menu entry is selected

```
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



6. Menu management (contd.)

```
void glutAttachMenu(int button)
void glutDetachMenu(int button)
```

button button to attach or detach a menu.

glutAttachMenu

attaches a mouse button for the current window to the identifier of the current menu

glutDetachMenu

detaches an attached button from the current window.

button should be one of the

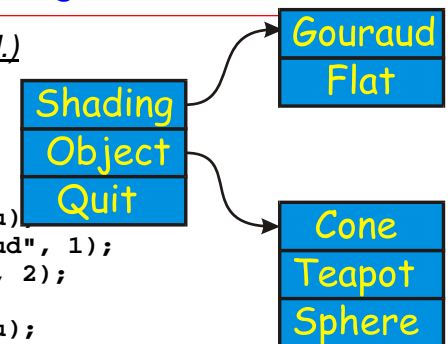
```
GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON,
GLUT_RIGHT_BUTTON
```



6. Menu management (contd.)

Example:

```
void MakeMenu(){
    int s1, s2;
    //create first submenu
    s1 = glutCreateMenu(Menu);
    glutAddMenuEntry("Gouraud", 1);
    glutAddMenuEntry("Flat", 2);
    //create second submenu
    s2 = glutCreateMenu(Menu);
    glutAddMenuEntry("Cone", 3);
    glutAddMenuEntry("Teapot", 4);
    glutAddMenuEntry("Sphere", 5);
    //create main menu and attach the others
    glutCreateMenu(Menu);
    glutAddSubMenu("Shading", s1);
    glutAddSubMenu("Object", s2);
    glutAddMenuEntry("Quit", 6);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```





```
void Menu(int val) //event handling
{
    static last=3; //default object is a cone
    switch (val) {
        case 1:glShadeModel(GL_FLAT);break;
        case 2:glShadeModel(GL_SMOOTH);break;
        case 3:last=3;break;
        case 4:last=4;break;
        case 5:last=5;break;
        case 6:exit(0);break;
    } // end of switch
    Render(last);
}

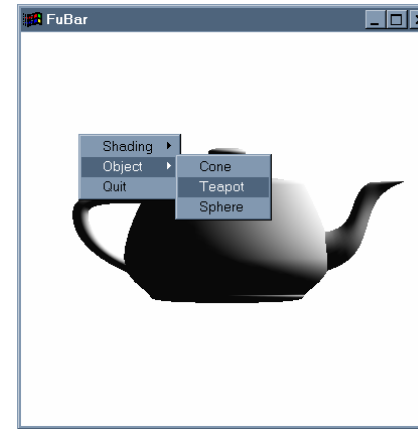
void Render(int val)
{
    switch (val) {
        case 3:glutSolidCone(0.3,0.8,20,20);break;
        case 4:glutSolidTeapot(0.5);break;
        case 5:glutSolidSphere(0.3,20,20);break;
    } // of switch
}
```

(C) Bedrich Benes

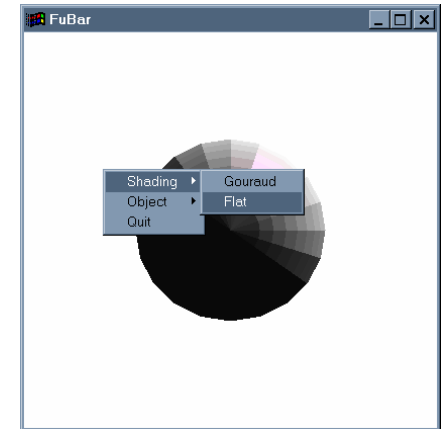
65



6. Menu management (contd.)



(C) Bedrich Benes



66



- they are pure OpenGL routines
- they do *not* generate display lists
- they generate normal vectors for appropriate lighting
- they do *not generate* texture coordinates (except for teapot)
- every routine has two versions
Solid and **Wire**
(only solid versions are described here)

(C) Bedrich Benes

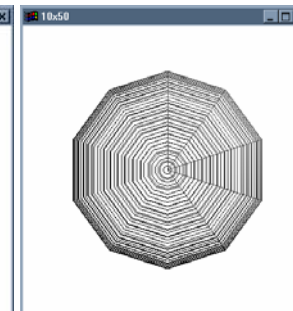
67



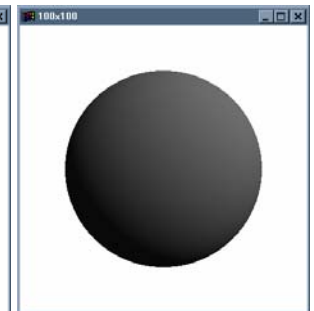
```
void glutSolidSphere(GLdouble radius,
                    GLint slices, GLint stacks)
```



5x10



10x50



100x100

(C) Bedrich Benes

68



```
void glutSolidCube(GLdouble size)
```

size length of each edge

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

- the base of the cone is placed at $Z = 0$
- the top at $Z = \text{height}$
- it is subdivided around the Z into slices
- and along the Z axis into stacks.



(C) Bedrich Benes



```
void glutSolidTorus(GLdouble inRad,
GLdouble outRad, GLint nsides, GLint rings)
```

```
void glutSolidDodecahedron(void)
```

```
void glutSolidOctahedron(void)
```



(C) Bedrich Benes

70



```
void glutSolidIcosahedron(void)
```

(20 sides)

```
void glutSolidTeapot(GLdouble size)
```

- surface normal and texture coordinates are generated
- the teapot is generated by OpenGL evaluators.

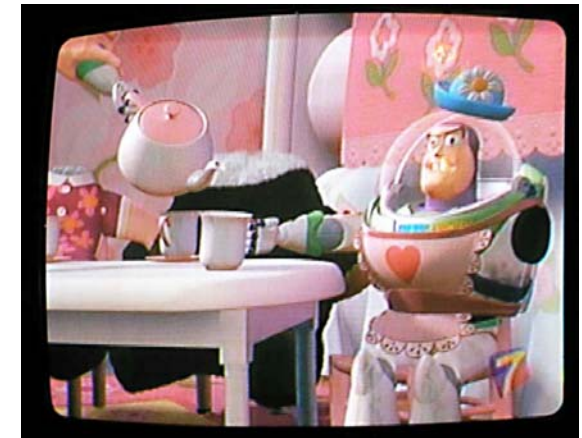


(C) Bedrich Benes

71



Why teapot?
It's a loooooong story....



(C) Bedrich Benes

72



- everything to display should be called from the callback of display
- do not call the display callback, use `glutPostRedisplay()`
- if you use the Idle callback to render, use Visibility callback to determine if the window really is visible
- do not forget to reshape by yourself! GLUT does not make it
- avoid use of the index color mode...



- carefully with the *current windows* or *current menus*.
- If you use more than one window or menu, use `glutSet*()` to determine where are you going to render etc.
- idle and timer callback work when menu is opened
- your program will run faster if you pass NULL to any not used callback e.g., `glutMotionCallback(NULL)`
- if you want to make an application that runs everywhere, check if the display mode you need is supported, i.e., call `glutGet(GLUT_DISPLAY_MODE_POSSIBLE)`



common info:

www.opengl.org/developers/documentation/glut/index.html?glut#first_hit

the GLUT Specification that has served as a basis for this document

on-line manual:

www.opengl.org/developers/documentation/glut/spec3/spec3.html

SIGGRAPH 2001 An Interactive Introduction To OpenGL Programming

www.opengl.org/developers/code/s2001/index.html