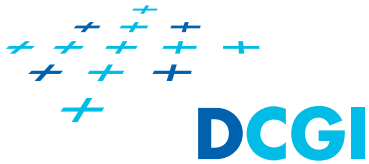


Programování grafiky GLUT

Petr Felkel

Katedra počítačové grafiky a interakce, ČVUT FEL

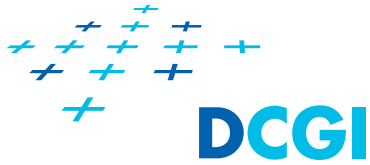
místnost KN:E-413 (Karlovo náměstí, budova E)

E-mail: felkel@fel.cvut.cz

S použitím materiálů Bohuslava Hudce, Jaroslava Sloupa a Vlastimila Havrana

Poslední změna: 26.2.2013

OpenGL Utility Toolkit (GLUT)

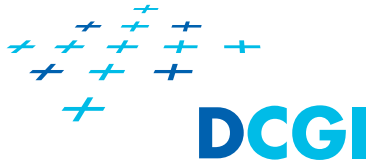


The OpenGL Utility Toolkit je **softwarové rozhraní** (**API**) **pro psaní programů v OpenGL nezávislé na platformě**

GLUT podporuje tuto funkcionalitu:

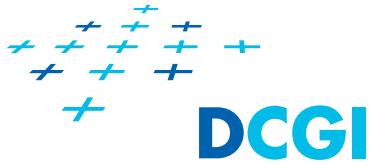
- *Správu oken (“windows”)*.
- *Zpracování zpětných událostí (“Callback* driven event processing”).
- Jednoduché kaskádové *pop-up menu*.
- Rutinu “*idle*” a časovače ”*timers*”.
- Rutiny pro různé objekty (“*solid and wire frame objects”)*.
- Podporu pro fonty (“*bitmap and stroke fonts”)*.
- Další funkce pro práci s okny (“*window management* functions”).

Typické použití knihovny GLUT



- Inicializuj GLUT a OpenGL
- Vytvoř okna a nastav jejich vlastnosti
- Vytvoř menu
- Nastav funkce zpětného volání ”callbacks”
  (alespoň jednu pro zobrazování obsahu okna)
- Inicializuj další části aplikace (načítání modelů, textur, …
- Spusť hlavní smyčku programu zobrazující okno (*main loop*)

Inicializace



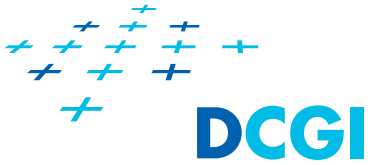

```
void glutInit(int *argc, char **argv);
```

argc *nemodifikovaná* proměnná argc z funkce main(...).

argv *nemodifikovaná* proměnná argv z funkce main(...).

- Inicializuje knihovnu GLUT a její napojení na správu oken
- Může byt ukončena s chybovou hláškou (pokud chybí podpora pro zobrazení oken s OpenGL nebo nejsou platné argumenty argc a argv)
- Zpracuje parametry na příkazové řádce (zpracuj a vyjmi nastavení pro GLUT z argv a argc)

Inicializace (pokrač.)



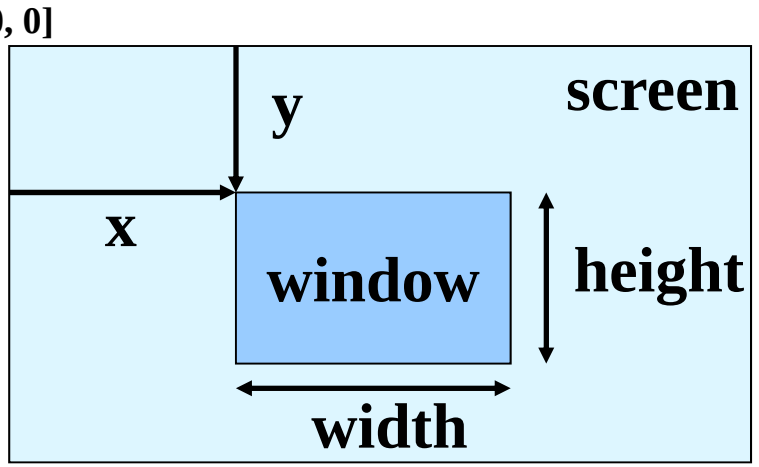

```
void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y);
```

width šířka okna v pixelech (iniciální hodnota 300).

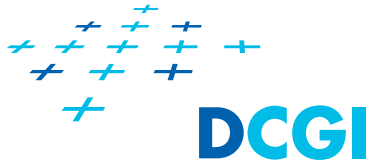
height výška okna v pixelech (iniciální hodnota 300).

x, y umístění okna na obrazovce (iniciální hodnoty x=y=-1, t.j, rozhodne okenní systém).

- je to pouze doporučení pro velikost okna i jeho umístění
- => ke zjištění skutečné polohy a velikosti okna je vhodné použít funkci zpětného volání *reshape* !



Inicializace (pokrač.)



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

mode režim zobrazování (viz pozdější přednášky)
- bitový OR masek pro GLUT display mode bit:

- **GLUT_RGBA** okno s barvami RGBA **(default),**
- GLUT_RGB alias pro GLUT RGBA,
- GLUT_INDEX okno s barevnými indexy - paleta,
- GLUT_SINGLE "a single buffered window" **(default),**
- **GLUT_DOUBLE** "double buffered window",
- GLUT_ACCUM okno s akumulačním bufferem,
- GLUT_ALPHA okno se složkou alpha v obrazové paměti,
- **GLUT_DEPTH** okno s pamětí hloubky "depth buffer",
- GLUT_STENCIL okno s pamětí šablony "stencil buffer".

Inicializace (pokrač.)

DCGI

single *versus* double buffer

buffermode.cpp

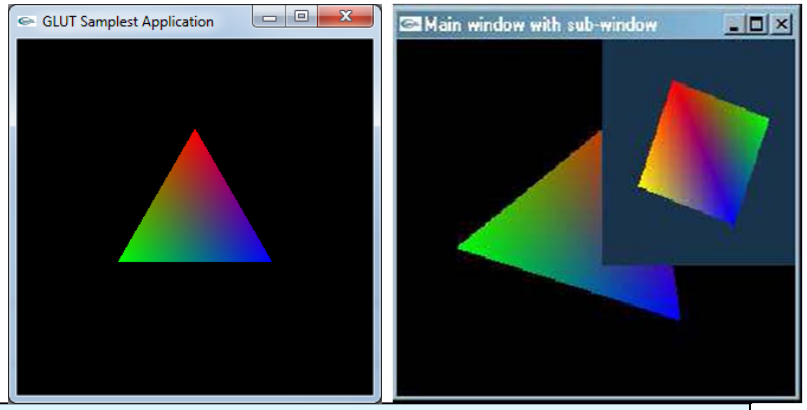
- single buffer mode
  - Obsah paměti je přímo zobrazován na obrazovce
  - Používá se pro statické scény
  - Není vhodný pro animace, neboť obrázky při zobrazování blikají

- **double buffer mode** (*rovněž "off-screen rendering"*)
  - Vhodný pro animované scény
  - Obrázek je vykreslován do druhé vrstvy obrazové paměti, která není zobrazena, neboť se zobrazuje první vrstva => na konci výpočtu obrázku jsou vrstvy přehozeny. Je zobrazena druhá vrstva a do první vrstvy se začne vykreslovat další snímek – použití příkazu glutSwapBuffers()

Inicializace (pokrač.)

```
int main(int argc, char **argv) {

    /* initialize GLUT and OpenGL */
    glutInit(&argc, argv);

    /* set display mode */
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );

   /* set initial window size */
    glutInitWindowSize(500, 500);

    …
}
```

Window management



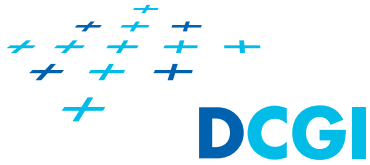
GLUT podporuje dva typy oken:

- **top-level windows** (samostatná okna)
- **sub-windows** (části oken, podokna)

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

- Vytvoříá samostatné okno označené jménem **name**
- Vrátí celočíselný identifikátor okna
- Identifikátor okna může být použit pro glutSetWindow()
- Do okna se nezačne vykreslovat dokud není vykonán příkaz glutMainLoop()

Window management (pokrač.)




```
int glutCreateSubWindow(
        int win, int x, int y, int width, int height);
```

win identifikátor rodičovského okna

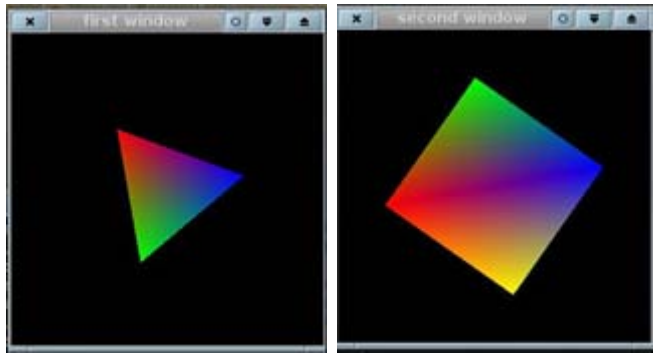
x, y umístění okna v pixelech vzhledem k rodičovskému oknu.

width šířka okna v pixelech.

height výška okna v pixelech.

- Vytvoří podokna v okně identifikovaném číslem **win** s danou šířkou a výškou na pozici **x** a **y**
- Podokna nelze ikonizovat
- Podokna lze libovolně vnořovat do sebe

Window management (pokrač.)

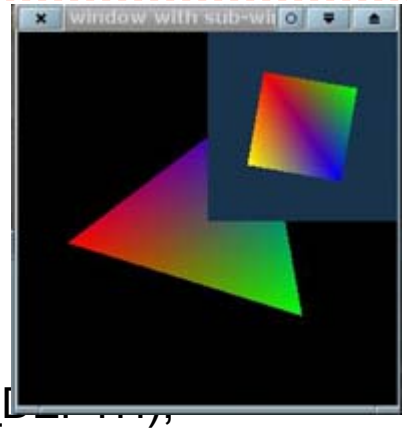


Příklad: dvě samostatná okna

```
int main(int argc, char **argv) {
    /* Initialize GLUT  */
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);

    /* Create first top-level window */
    glutInitWindowPosition(5, 5);
    glutInitWindowSize(300, 300);
    int winId1 = glutCreateWindow("First window");
    …
    /* Create second top-level window */
    glutInitWindowPosition(310, 5);
    int winId2 = glutCreateWindow("Second window");
    …
}
```

twowindows.cpp

Window management (pokrač.)

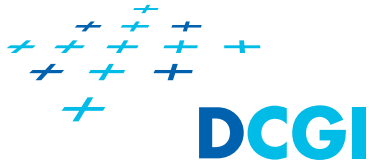


Příklad: hlavní okno s jedním podoknem

```
int main(int argc, char **argv) {
    /* Initialize GLUT  */
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_D…);

    /* Create top-level window */
    glutInitWindowPosition(5, 5);
    glutInitWindowSize(300, 300);
    int winId = glutCreateWindow("Main window with sub-window");
    …
    /* Create sub-window */
    int subWinId = glutCreateSubWindow(winId, 150, 0, 150, 150);
    …
}
```

subwindow.cpp

Window management (pokrač.)



Další užitečné funkce:

```
void glutSetWindow(int winId);
int glutGetWindow(void);
```

```
void glutDestroyWindow(int winId);
```

```
void glutReshapeWindow(int width, int height);
```

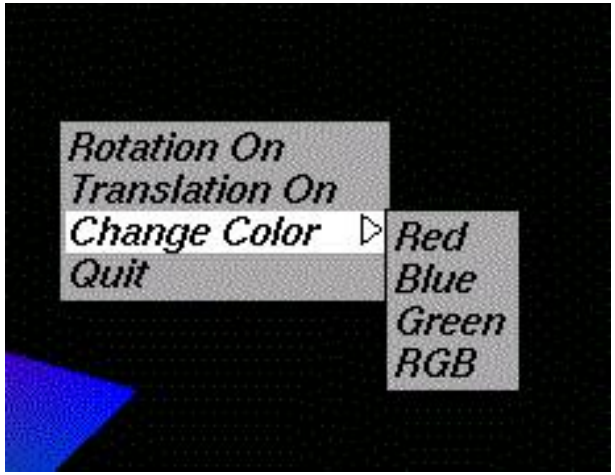
```
void glutSetWindowTitle(char *name);
```

```
void glutFullScreen(void);
```

Menu management (pokrač.)

GLUT podporuje jednoduchá kaskádová menu

- Umožňují výběr z nabídky



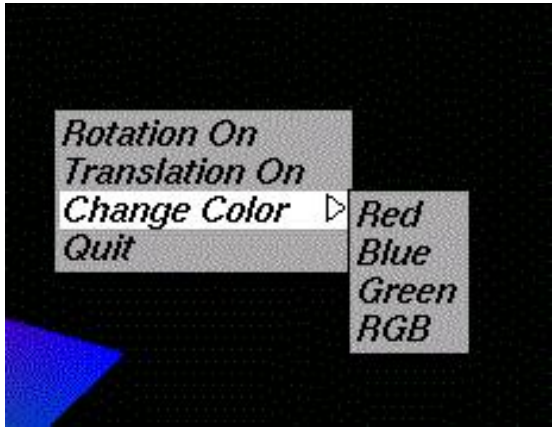
- Mají jednoduchou a minimalistickou funkcionalitu (není vhodná pro implementaci složitých menu)

- Není možné vytvářet, modifikovat nebo mazat menu nebo submenu během jejich používání (jsou-Ii rozbalena)

Menu management (pokrač.)

int **glutCreateMenu**(void (***menuFunc**)(int value));

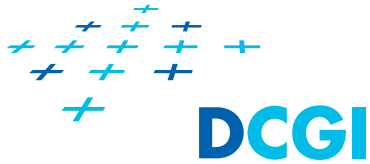
menuFunc funkce zpětného volání “callback function”, která je zavolána při výběru položky menu. Parametr “value” je předán funkci zpětného volání.




```
void menuFunc(int value) {
  … /* actions depending on the passed value */
}
```

- Vytvoří nové pop-up menu a vrátí jeho jednoduchý celočíselný identifikátor

Menu management (pokrač.)



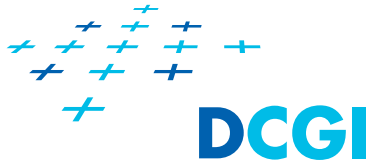
void **glutAddMenuEntry**(char ***name**, int **value**);

name řetězec ASCII znaků, který se zobrazí v položce menu.

value hodnota, která je předána funkci zpětného volání po výbrání položky v menu

- Přidá položku do menu na konec již existujících položek
- Řetězec znaků **name** bude zobrazen v položce menu
- Pokud je tato položka menu vybrána uživatelem, je zavolána funkce zpětného volání s parametrem **value**

Menu management (pokrač.)



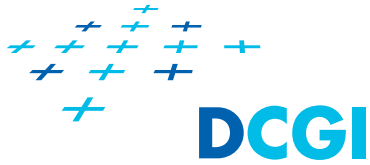
void **glutAddSubMenu**(char ***name**, int **menu**);

name řetězec ASCII znaků v položce, po jejímž výběru se zobrazí podmenu

menu identifikátor menu, které se zobrazí jako kaskádové podmenu

- Přidá spouštěč podmenu na konec již existujících položek aktuálního menu
- Řetězec znaků **name** se zobrazí v položce menu, ze které je zobrazeno podmenu
- Kliknutím na text spouštěče podmenu se podmenu vyvolá a lze v něm vybírat mezi jeho položkami

Menu management (pokrač.)



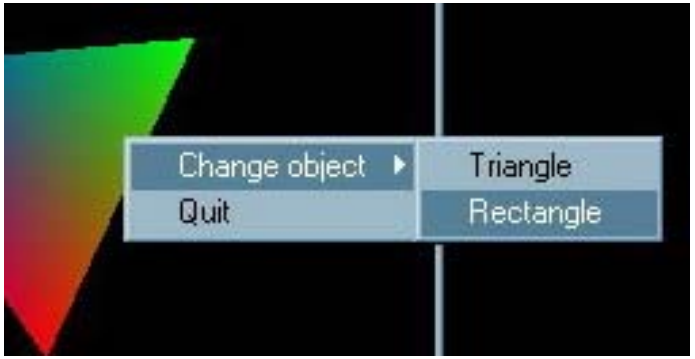

```
void glutAttachMenu(int button);

void glutDetachMenu(int button);
```

button přiřazení a odpojení tlačítka, kterým se bude menu vyvolávat

- glutAttachMenu přiřadí menu k tlačítku myši (**button)** pro aktuální okno. Menu se zobrazí při stisku tohoto tlačítka
- **glutDetachMenu** odpojí menu od přiřazeného tlačítka myši
- Tlačítko je buď **GLUT_LEFT_BUTTON**, **GLUT_MIDDLE_BUTTON** a nebo **GLUT_RIGHT BUTTON**

Menu management (pokrač.)

Příklad: menu s pod-menu a jednou položku




```
/* at first create a sub-menu */
int menuId = glutCreateMenu(myMenu);
glutAddMenuEntry("Triangle", 1);
glutAddMenuEntry("Rectangle", 2);

/* create a main menu */
glutCreateMenu(myMenu);
glutAddSubMenu("Change object", menuId);
glutAddMenuEntry("Quit", 3);

/* menu will be invoked by left mouse button */
glutAttachMenu(GLUT_LEFT_BUTTON);

object = OBJ_TRIANGLE;
```

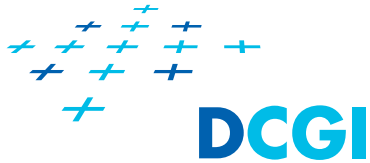
menu.cpp

Menu management (pokrač.)

```
/* this function performs an action for the selected menu item */
void myMenu(int menuEntryId) {

    switch(menuEntryId) {
        case 1:             /* menu entry "Triangle" */
          object = OBJ_TRIANGLE;
         break;
        case 2:             /* menu entry "Rectangle" */
          object = OBJ_RECTANGLE;
          break;
        case 3:             /* menu entry "Quit" */
          exit(0);
    }
}
```

menu.cpp

Zpracování událostí

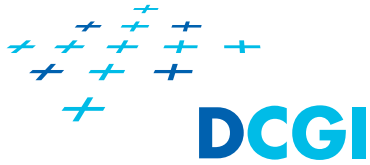


Systémy pro zobrazování oken jsou založeny na událostech:

- otevření okna (window)
- překreslení okna
- změna velikosti okna
- pohyb myši
- stisk tlačítka
- zavření okna
- výběr položky v menu
- a jiné.

Zpětné volání (“Callback”) je funkce, která je registrována systémem a je vyvolána při výskytu události.

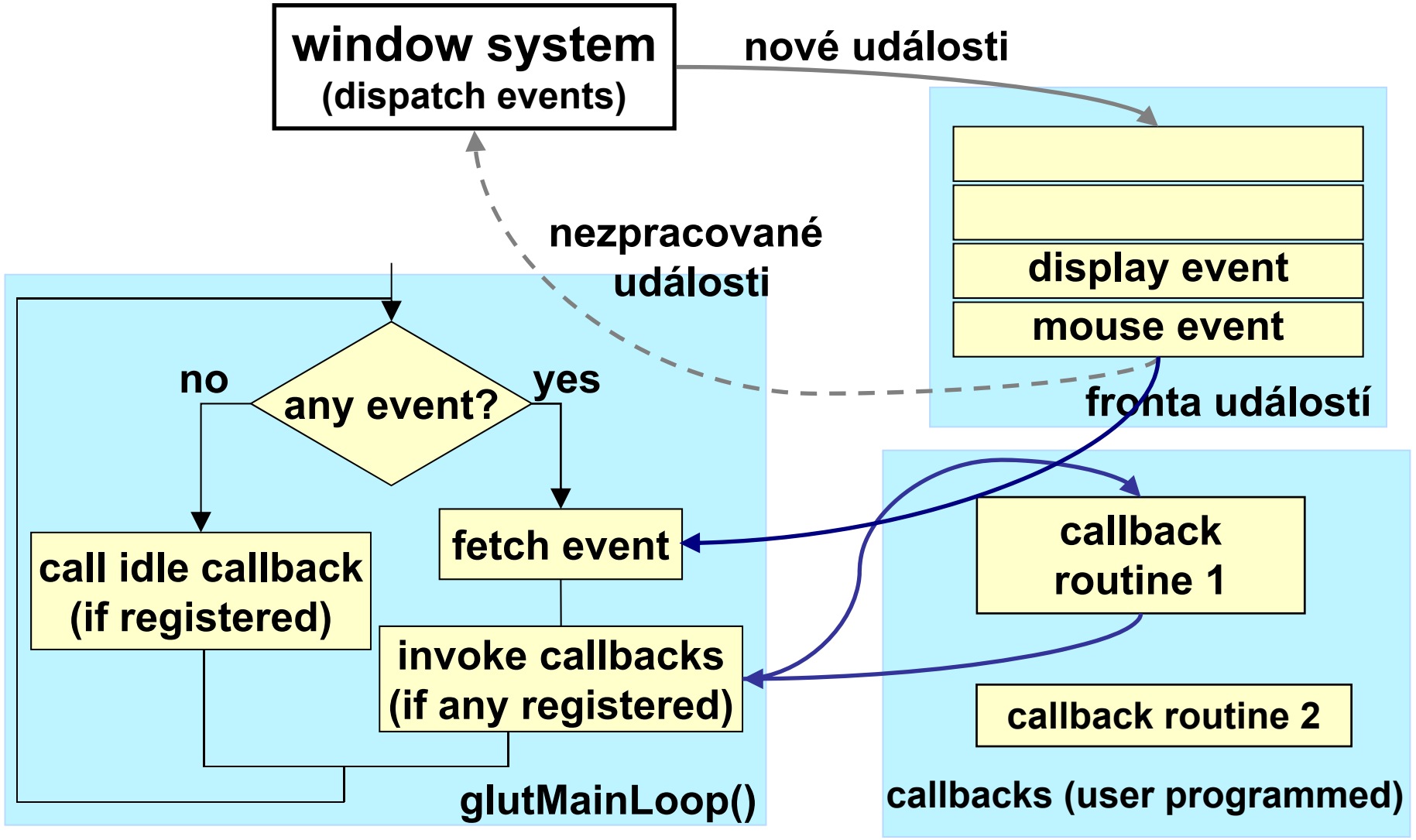
Zpracování událostí – hlavní smyčka



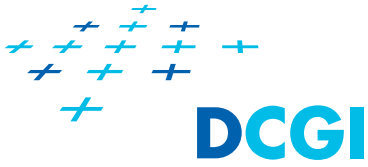
void **glutMainLoop**(void);

- Napsána jako poslední funkce ve funkci **main**, která spustí inicializaci GLUTu (vytvoření oken a menu)
- Spustí hlavní smyčku zpracování
- V tradičním GLUTu je glutMainLoop() vyvolána pouze jednou, v implementaci freeglut je možné ji spustit opakovaně
- Zpracování událostí je založeno na zpětném volání funkcí vyvolaných při výskutu událostí
- Funkce zpětného volání (**callbacks**) musí být registrovány

Registrace funkcí zpětného volání



Registrace funkcí zpětného volání – překreslení okna




```
void glutDisplayFunc(void (*displayFunc)(void));
```

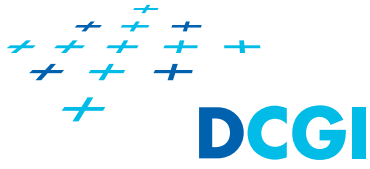
displayFunc tato funkce (callback function) je vyvolána, pokud je nutné překreslit okno

```
void displayFunc(void) {
  /* Clear frame buffer */
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  /* code for scene rendering should appear here */

   glutSwapBuffers(); /* finally, if the double buffering is enabled */
}
```

- Funkce zpětného volání displayFunc **musí být** registrována pro všechna okna
- Nikdy nevolejte funkci displayFunc přímo!
- Je možné ji vyvolat přes volání funkce:

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

Registrace funkcí zpětného volání – změna velikosti okna




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

- Nastaví funkci zpětného volání pro aktuální okno při změně jeho velikosti

```
void reshapeFunc(int width, int height) {
    glViewport(0, 0, width, height); /* setup viewport transformation */
    /* Setup projection matrix  */
}
```

- Funkce **reshapeFunc** je vyvolána, kdykoliv dojde ke změně velikosti okna a také před prvním zavoláním zpětného volání funkce zobrazení **displayFunc**
- Funkci je předána nová velikost okna, t.j. šířka a výška okna v pixelech
- Při změně velikosti hlavního okna, nejsou změněny velikosti podoken => ve funkci **reshapeFunc** hlavního okna se musí každé podokno nastavit:

```
glutSetWindow( subWin );
glutPositionWindow(width/2, 0);          /* new subWin position */
glutReshapeWindow(width/2, height/2); /* new subWin size */
```

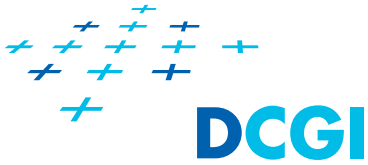
Registrace funkcí zpětného volání – standardní klávesy


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

```
void keyboardFunc(unsigned char key, int x, int y) {
    switch (key) {
        case 'q':      /* ‘q’ or ‘Q’ key pressed => terminate application */
        case 'Q':
            exit (0);
    }
}
```

- Stisk klávesy vyvolá zpětné volání s předáním ASCII znaku (parametr **key**)
- Proměnné **x** and **y** jsou pozice kurzoru myši
- Během zpracovávání zpětného volání pro obsluhu klávesnice je možné zavolat funkci int **glutGetModifiers(void)** a zjistit, které funkční klávesy byly stisknuty (GLUT_ACTIVE_SHIFT, GLUT_ACTIVE_CTRL or GLUT_ACTIVE_ALT)

Registrace funkcí zpětného volání – funkční klávesy

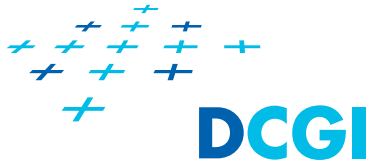



```
void glutSpecialFunc(
        void (*keyboardSpecialFunc)(int key, int x, int y));
```

```
void keyboardSpecialFunc(int key, int x, int y) {
    switch (key) {
        case GLUT_KEY_LEFT:     /* perform some action */
            break;
        case GLUT_KEY_RIGHT:    /* perform some action */
            break;
    }
}
```

- Funkce zpětného volání pro funkční klávesy a šipky
- Parametr **key** nabývá hodnoty jedné z konstant GLUT_KEY_* podle toho, která speciální klávesa (například GLUT_KEY_UP, GLUT_KEY_F1, or GLUT_KEY_PAGE_UP) byla zároveň stisknuta.

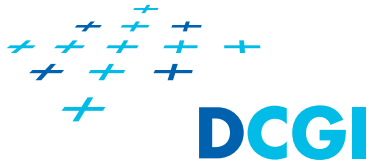
Registrace funkcí zpětného volání – stisk tlačítek myši



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

- Vyvolána, pokud uživatel stiskne nebo uvolní tlačítka myši *(“mouse buttons”)* s kurzorem myši umístěným v okně – každý stisk či uvolnění tlačítka vyvolá funkci zpětného volání (“mouse callback”)
- Parameter **button** nabývá jednu z možností GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON
- Parametr **state** je buď GLUT_UP nebo GLUT_DOWN a indikuje, zda došlo ke stisku či uvolnění tlačítka myši

Registrace funkcí zpětného volání – pohyb myši



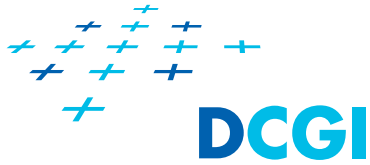
void **glutMotionFunc**(void (***motionFunc**)(int **x**, int **y**));

- Tato funkce zpětného volání je vyvolána, pokud dojde k pohybu myši v okně a je současně stisknuto jedno či více tlačítek myši

void **glutPassiveMotionFunc**(void (***motionFunc**)(int **x**, int **y**));

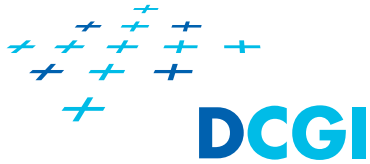
- Funkce je vyvolána pokud není stisknuto žádné tlačítko při pohybu myši nad oknem

Registrace funkcí zpětného volání (pokrač.)




```
void mouseFunc(int button, int state, int x, int y) {
    /* which button was pressed ? */
    switch(button) {
        case GLUT_LEFT_BUTTON:     /* some action */ break;
        case GLUT_MIDDLE_BUTTON: /* some action */ break;
        case GLUT_RIGHT_BUTTON:   /* some action */ break;
    }
    /* check button state – pressed or released ? */
    switch(state) {
        case GLUT_UP:       /* some action */ break;
        case GLUT_DOWN: /* some action */ break;
    }
}
```

Registrace funkcí zpětného volání (pokrač.)



void **glutIdleFunc**(void (***idleFunc**)(void));

- Nastaví zpětné volání na funkci **idleFunc**
- Pokud je nastaveno, funkce se opakovaně volá tehdy, nejsou-li ve frontě žádné události ke zpracování

  => **program může provádět výpočet, například animace** ☺

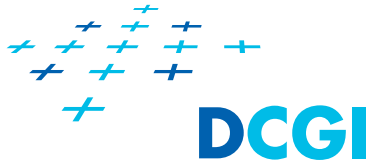
- Provádění rychlosti výpočtu je závislé na poměru rychlosti zpracování operací na CPU/GPU a na rychlosti zpracování vlastního zobrazování (rychlejší CPU –> rychlejší animace)
- Množství výpočtu a čas pro výpočet zobrazení ve funkci idleFunc musí být minimalizováno aby zůstala zachována interaktivita programu (složité zobrazování –> pomalá animace)
- Obecně, by ve funkci **idleFunc** měl být počítán maximálně **jeden snímek pro výpočet zobrazení**

Registrace funkcí zpětného volání (pokrač.)

```
void idleFunc(void) {
    spin += spinStep;          /* set new angle for rotation */
    glutPostRedisplay ();      /* redraw window */
}
int main(int argc, char **argv) {
    …
    glutCreateWindow("Events example");
    glutDisplayFunc(displayFunc);   /* set callbacks */
    glutReshapeFunc(reshapeFunc);
    glutKeyboardFunc(keyboardFunc);
    glutMouseFunc(mouseFunc);
    glutIdleFunc(idleFunc);
    glutMainLoop ();           /* finally, enter event loop */
}
```

events.cpp

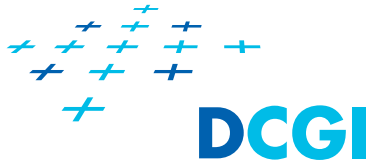
Registrace funkcí zpětného volání (pokrač.)




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

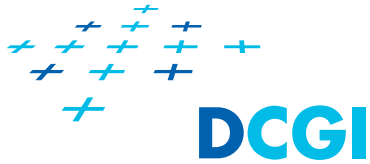
- Registruje funkci zpětného volání pro časový čítač ***timerFunc***, která bude jednou vyvolána nejméně po ***msecs*** millisekundách (naplánuje jednu událost ve frontě událostí)
- GLUT se snaží vyvolat funkci ***timerFunc*** co nejdříve po uplynutí uvedeného intervalu.
- Je možné najednou registrovat několik funkcí zpětného volání s různými hodnotami časové prodlevy a hodnoty **value**
- Při vyvolání funkce ***timerFunc*** je do parametru ***value*** předána hodnota ***value***, která byla zadána při registraci funkce zpětného volání. Tato hodnota slouží k identifikaci zdroje časování.
- Pro animace musí funkce zpětného volání ***timerFunc*** registrovat znovu sama sebe pomocí ***glutTimerFunc***, aby naplánovala vyvolání další události

Registrace funkcí zpětného volání (pokrač.)




```
void timerFunc(int id)          // USE THIS to SAVE PROCESSOR TIME
{   // possible processing of id value
    glutTimerFunc(33, timerFunc, 0);        /* register new animation step */
    spin += spinStep;                       /* set new angle for rotation */
    glutPostRedisplay();                    /* redraw window */
}
int main(int argc, char **argv) {
    …
    glutCreateWindow("Events example");
    glutDisplayFunc(displayFunc);   /* set callbacks */
    glutReshapeFunc(reshapeFunc);
    glutKeyboardFunc(keyboardFunc);
    glutMouseFunc(mouseFunc);
    glutTimerFunc(33, timerFunc, 0);
    glutMainLoop ();   /* finally, enter event loop */
}
```

Funkce navíc ve freeGLUTu



FreeGLUT

- Zachovává funkce GLUTu
- Rozvíjí se (OpenSource)
- Přidává obsluhu kolečka myši, multisampling, nastavení kontextu OpenGL a ovládání hlavní smyčky programu
- #include <GL/freeglut.h>
  - interně funKce rozděleny na
  - #include "freeglut_std.h„ - původní funkce GLUTu
  - #include "freeglut_ext.h„ - nové funkce freeGLUTu

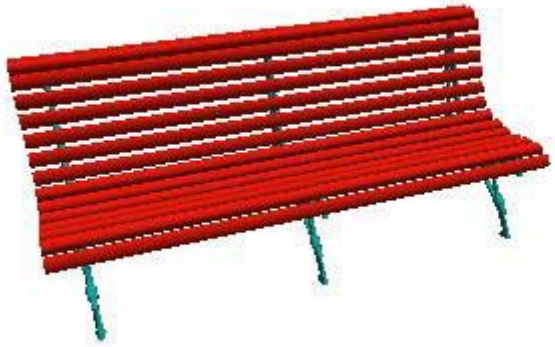
Multisampling

Vyhlazení scény grafickou kartou – více vzorků

```
glutInitDisplayMode(GLUT_DEPTH |
                    GLUT_DOUBLE |
                    GLUT_RGBA |
                    GLUT_MULTISAMPLE);  // nastavení v GLUT
```

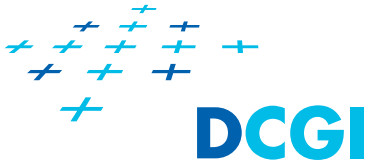
Zapnutí

```
glEnable(GL_MULTISAMPLE)
```



[lighthouse3d]

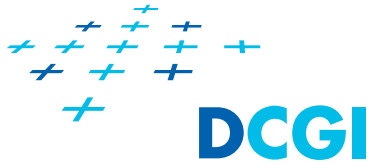
Celá obrazovka - *Game Mode*




```
char mode_string[20];
sprintf(mode_string,"%dx%d:32@60",
        glutGet(GLUT_SCREEN_WIDTH),
        glutGet(GLUT_SCREEN_HEIGHT));
glutGameModeString(mode_string);
...
// When leaving GameMode freeglut requires
// setting the window again
if (glutGameModeGet(GLUT_GAME_MODE_ACTIVE) != 0) {
        glutLeaveGameMode();
        glutSetWindow(mainWindow);
}
```

[lighthouse3d]

Obsluha kolečka myši -*MouseWheel*



Kolečku zaregistrujeme události:

```
glutMouseWheelFunc(myMouseWheel Func)
```

Funkce zpětného volání bude mít parametry

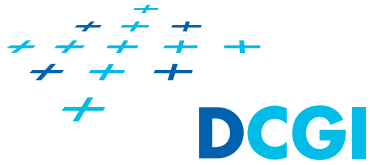
```
void myMouseWheel (int wheel, int direction, int x, int y)
```

- wheel: číslo kolečka myši – jediné kolečko má hodnotu 0.
- Směr: a +/- 1 Hodnota a udává, o kolik je kolečko otočilo
- x, y: souřadnice kurzoru v okně

- Funkce bude zavolána při pootočení kolečka
- Při stisknutí se volá glutMousePressed()

[lighthouse3d]

Obsluha hlavní smyčky programu



Nastavení akce při zavření okna:

```
glutSetOption(   GLUT_ACTION_ON_WINDOW_CLOSE, hodnota );
```

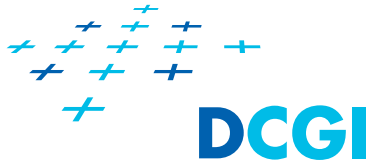
hodnota je

- GLUT_ACTION_GLUTMAINLOOP_RETURNS
  - opustí smyčku
  - pokračuje za glutMainLoop()
- GLUT_ACTION_CONTINUE_EXECUTION
  - při více oknech ignoruje požadavek zavření okna
  - při jediném okně jako první varianta

```
glutSetOption(   GLUT_ACTION_ON_WINDOW_CLOSE,
                 GLUT_ACTION_GLUTMAINLOOP_RETURNS);
glutMainLoop();
printf("I'm out\n");  // místo např. pro úklid datových struktur
```

[lighthouse3d]

Obsluha hlavní smyčky programu




```
glutLeaveMainLoop();
```

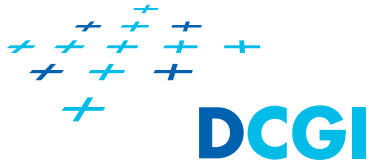
- Měla by okamžitě opustit smyčku, ale
- Smyčka skončí až po vyprázdnění fronty událostí
- Vrátí se za glutMainLoop();
- Vhodná např. pro ošetření klávesy Esc

poznámek

```
void processKeys(unsigned char key, int xx, int yy)
{
    switch(key) {
    case 27: // QUIT
            glutLeaveMainLoop();   // namísto exit(0);
   }
}
```

[lighthouse3d]

Callback funkce po uzavření okna

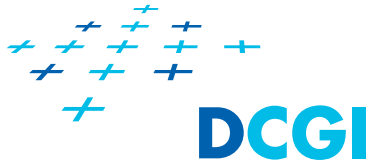



```
glutCloseFunc ( void( *callback )( void ))
```

- Ideální funkce pro uvolnění zdrojů

[lighthouse3d]

Zpracování jedné události



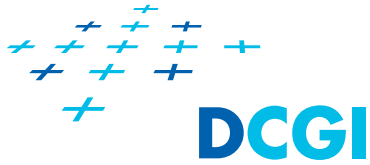

```
glutMainLoopEvent();
```

- Zpracuje jednu iteraci z fronty událostí
- Iterace odpovídá vyprázdnění fronty událostí
- Není vhodné ji kombinovat s funkcí Idle()

```
while (!loopExit)
{
        ...
        glutMainLoopEvent();   // zpracuje aktuální frontu událostí
        renderScene();         // namísto Idle()
        ...
}
```

[lighthouse3d]

Nastavení kontextu OpenGL

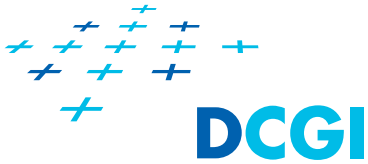



```
glutInitContextVersion (int majorVersion, int minorVersion)
glutInitContextProfile ( int profile)
glutInitContextFlags ( GLuint flags)
```

- Nastaví hodnoty interní struktury pro definování OpenGL kontextu
- Profil – GLUT_CORE_PROFILE nebo
  – GLUT_COMPATIBILITY_PROFILE

```
// příklad ze cvičení pgr
glutInitContextVersion( 3, 1 );
glutInitContextFlags(GLUT_FORWARD_COMPATIBLE);
```

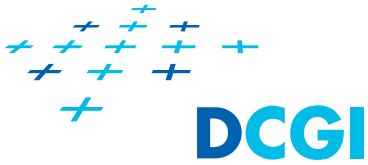
Zpracování chyb v OpenGL



GLenum **glGetError(**void**);**

- OpenGL detekuje jen minimum z možných chyb
- Kontrola všeho by byla příliš časově náročná
- Zapamatuje si první chybu, která nastala
- Každá chyba má svůj vlastní číselný kód a jemu přiřazenou symbolickou konstantu:
  - **GL_NO_ERROR** *no problem, OK*
  - **GL_INVALID_ENUM** *enum value out of range.*
  - **GL_INVALID_VALUE** *numeric argument is out of range*
  - **GL_INVALID_OPERATION** *illegal operation in current state*
  - **GL_INVALID_FRAMEBUFFER_OPERATION** *offending command for current state*
  - **GL_OUT_OF_MEMORY** *not enough memory to execute the command*

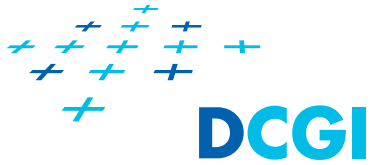
Rozšíření debug_output



- Umožňuje zaregistrovat funkci zpětného volání (*callback*), která je vyvolána při chybě
- GL_ARB_debug_output
  - dostupná již od verze 1.1
  - od OpenGL verze 4.3 je součástí core profile
- Nahrazuje glGetError()
  - musel se explicitně zavolat
- Pracuje jen v režimu debug

```
glutInitContextFlags( GLUT_FORWARD_COMPATIBLE
                    | GLUT_DEBUG);
```

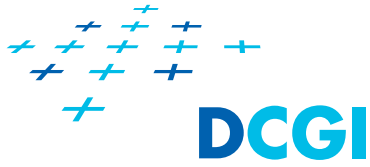
Příklad funkce pro obsluhu chyby




```
if(glDebugMessageCallbackARB){
                        cout << "Register OpenGL debug callback " << endl;
                        glEnable(GL_DEBUG_OUTPUT_SYNCHRONOUS_ARB);
                        glDebugMessageCallbackARB(openglCallbackFunction, nullptr);
```

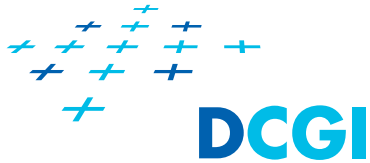
```
void APIENTRY openglCallbackFunction(
                                GLenum source,          // zdroj chyby (API, okna, překlad sh.)
                                GLenum type,            // typ chyby (error, deprecated,…)
                                GLuint id,              // číslo chyby
                                GLenum severity,        // důležitost chyby (low-medium-high)
                                GLsizei length,
                                const GLchar* message, // popis chyby
                                void* userParam)        // ukazatel zadaný ukazatel na data
{
        cout << "---------------------opengl-callback-start------------" << endl;
        cout << "message: "<< message << endl;
        …
```

Příklad funkce pro obsluhu chyby



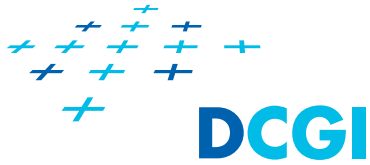

```
cout << "type: ";
switch (type) {
case GL_DEBUG_TYPE_ERROR:
        cout << "ERROR";
        break;
case GL_DEBUG_TYPE_DEPRECATED_BEHAVIOR:
        cout << "DEPRECATED_BEHAVIOR";
        break;
case GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR:
        cout << "UNDEFINED_BEHAVIOR";
        break;
case GL_DEBUG_TYPE_PORTABILITY:
        cout << "PORTABILITY";
        break;
case GL_DEBUG_TYPE_PERFORMANCE:
        cout << "PERFORMANCE";
        break;
case GL_DEBUG_TYPE_OTHER:
        cout << "OTHER";
        break;
}   cout << endl;
```

Příklad funkce pro obsluhu chyby




```
        cout << "id: "<<id << endl;

        cout << "severity: ";
        switch (severity){
        case GL_DEBUG_SEVERITY_LOW:
                cout << "LOW";
                break;
        case GL_DEBUG_SEVERITY_MEDIUM:
                cout << "MEDIUM";
                break;
        case GL_DEBUG_SEVERITY_HIGH:
                cout << "HIGH";
                break;
        }
        cout << endl;

cout << "---------------------opengl-callback-end--------------" << endl;
}
```

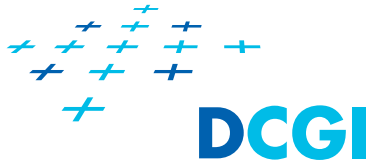
Řízení hlášení chyb




```
void glDebugMessageControl (
        GLenum          source,     // zdroj chyby (API, okna, překlad sh.)
        GLenum          type,       // typ chyby (error, deprecated,…)
        GLenum          severity,   // důležitost chyby (low-medium-high)
        GLsizei         count,      // délka po¨le s čísly chyb
        const GLuint    *ids,       // pole s čísly chyb
        GLboolean       enabled);   // GL_TRUE / GL_FALSE – povoluje hlášení
```

- Povolí či zakáže hlášení vyjmenovaných chyb

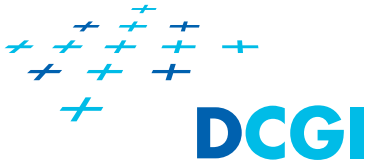
Příklad funkce pro obsluhu chyby




```
void glDebugMessageInsert (
                                GLenum source,          // zdroj chyby (API, okna, překlad sh.)
                                GLenum type,            // typ chyby (error, deprecated,…)
                                GLuint id,              // číslo chyby
                                GLenum severity,        // důležitost chyby (low-medium-high)
                                GLsizei length,
                                const GLchar* message); // popis chyby
```

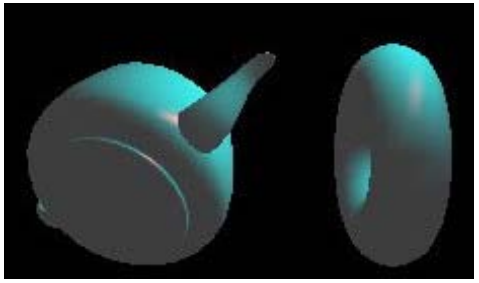
- Vsune aplikací vygenerovanou chybu do fronty chyb

Zobrazení geometrických objektů



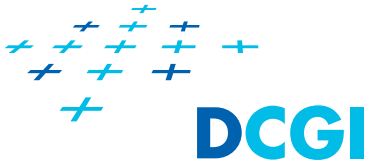
GLUT umožňuje vykreslení řady standardních geometrických objektů ve 3D, nicméně, to je zastaralé, využívá funkce staré verze OpenGL, nepoužívat!

- Funkce negenerují tzv. display listy
- Funkce generují normály pro pro osvětlování, ale žádné souřadnice pro teturování (kromě “UTAH teapot”)
- Každý objekt má dvě verze
  - **solid** glutSolidSphere, glutSolidTeapot, …
  - **wireframe** glutWireSphere, glutWireTeapot, …



shapes.cpp

Zajímavé odkazy



- **Root.cz. Tvorba přenositelných grafických aplikací využívajících knihovnu GLUT.**
  http://www.root.cz/serialy/tvorba-prenositelnych-grafickych-aplikaci-vyuzivajicich-knihovnu-glut/
- **GLUT FAQ**
  http://www.opengl.org/resources/libraries/glut/faq/
- **Stetten George and Crawford Korin: "*GlutMaster version 0.3*" (C++ wrapper)**
  http://www.stetten.com/george/glutmaster/glutmaster.html
- **GLUT Tutorial (lighthouse):**
  http://www.lighthouse3d.com/opengl/glut/
- Glut and FreeGLUT (lighthouse tutoriál)
  http://www.lighthouse3d.com/cg-topics/glut-and-freeglut/
- **FreeGlut**
  http://freeglut.sourceforge.net/