Agenda

- General rules for the course
- General info on the libraries
- GLUT
- OpenGL
- GLUI
- Details about GLUT Functions – Probably we will not cover this part
General Rules for the Course

• **Newsgroup**: You **must** follow the newsgroup on daily basis.

• **Project Groups**: Exactly 2 people

• **Cheating**: In case of cheating, all parts involved (source(s) and receiver(s)) get zero. We use automated tools for cheat detection.
Copy-Paste Issue

• There are several code samples on the internet, but we strongly advise you not to make copy-paste from such resources.

• Because it will decrease the knowledge that you can gain from this course. Moreover you can fail because of this…
What is OpenGL?

• OpenGL
  – A hardware and operating system independent graphics API specification.
  – Also windows system independent
  – Many vendors provide implementations of this specification for a variety of hardware platforms.
  – Bindings exist primarily for the C, Fortran and Ada programming languages.

• “A software interface to graphics hardware”

• (Low Level) Graphics rendering API
General Info on the Libraries

- OpenGL (Open Graphics Library)
- GLUT (Graphics Library Utility Toolkit)
- GLU (Graphics Utility Library)
- GLUI (User Interface Library)
OpenGL/GLUT/GLU/GLUI

• **OpenGL** is the “core” library that is platform independent.

• **GLUT** is an auxiliary library that handles window creation, OS system calls (mouse buttons, movement, keyboard, etc), callbacks.

• **GLU** is an auxiliary library that handles a variety of graphics accessory functions. It uses only GL functions but it contains code for common geometric objects, such as spheres, that users prefer not to write repeatedly.

• **GLUI** is a GUI manager for GLUT.
Preliminaries for the libraries

- Header Files
  
  ```c
  #include <GL/gl.h>
  #include <GL/glu.h>
  #include <GL/glut.h>
  #include <glui.h>
  ```
General Info on the Libraries

• About setting up the libraries on linux and windows, most of the things you need are on the course website under Downloads section.

• I will make a post on the course newsgroup with some brief information

• There are many system dependent issues at this point.

• For more info on this issue,
  – You may make some internet search
  – Post a message to metu.ceng.unix or metu.ceng.mswindows
GLUT

- GLUT is a window-system-independent toolkit; it provides tools to assist with event and windows management.

- GLUT supports
  - Multiple windows for OpenGL rendering.
  - Callback driven event processing.
  - Sophisticated input devices.
  - An “idle” routine and timers.
  - A simple, cascading pop-up menu facility.
  - Utility routines to generate various solid and wire frame objects.
  - Support for bitmap and stroke fonts.
  - Miscellaneous window management functions, including managing overlays.
GLUT and Event-Driven Programming

• Window-based graphic systems
• Event-driven programs
  – “do this whenever that happens” instead of “do this, then do this, then do this”
  – Wait until an event happens and then execute some pre-defined functions according to the user’s input
• Events
  – Click of a mouse button
  – Movement of the mouse cursor
  – Keyboard button being pressed
  – Window resize
• Event-queue
GLUT Basics

• Application Structure
  – Configure and open window
  – Initialize OpenGL state
  – Register input callback functions
    • render
    • resize
    • input: keyboard, mouse, etc.
  – Enter event processing loop
Sample Program

```c
#include <GL/glut.h>
#include <GL/gl.h>

void main(int argc, char** argv)
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow( "Simple" );
    init();
    glutDisplayFunc( display );
    glutKeyboardFunc( key );
    glutMainLoop();
}
```
Sample Program

```c
#include <GL/glut.h>
#include <GL/gl.h>

void main(int argc, char** argv)
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow( "Simple" );
    init();
    glutDisplayFunc( display );
    glutKeyboardFunc( key );
    glutMainLoop();
}
```

Specify the display

Mode - RGB or color

Index, single or double

Buffer
Sample Program

```c
#include <GL/glut.h>
#include <GL/gl.h>

void main(int argc, char** argv)
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow( "Simple" );
    init();
    glutDisplayFunc( display );
    glutKeyboardFunc( key );
    glutMainLoop();
}
```

Create a window Named “simple” with resolution 500 x 500
Sample Program

#include <GL/glut.h>
#include <GL/gl.h>

void main(int argc, char** argv)
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow( "Simple" );
    init();
    glutDisplayFunc( display );
    glutKeyboardFunc( key );
    glutMainLoop();
}
Sample Program

```c
#include <GL/glut.h>
#include <GL/gl.h>

void main(int argc, char** argv)
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutInitWindowSize( 500,500 );
    glutCreateWindow( "Simple" );
    init();
    glutDisplayFunc( display );
    glutKeyboardFunc(key);
    glutMainLoop();
}
```

Register your callback functions
glutMainLoop()

```c
#include <GL/glut.h>
#include <GL/gl.h>

int main(int argc, char** argv)
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode(mode);
    glutInitWindowSize(500,500);
    glutCreateWindow("Simple");
    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glutMainLoop();
}
```

The program goes into an infinite loop waiting for events
OpenGL Initialization

- Set up whatever state you’re going to use

```c
void init( void )
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-10, 10, -10, 10, -10, 20);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    /*glEnable( GL_LIGHT0 );
    glEnable( GL_LIGHTING );
    glEnable( GL_DEPTH_TEST );*/
}
```
GLUT Callback Functions

- **Callback function**: A programmer specified routine that can be registered to be called in response to a specific type of event (when an event happens)
  - Window resize or redraw
  - User input (mouse, keyboard)
  - Animation (render many frames)

- “Register” callbacks with GLUT
  - `glutDisplayFunc( my_display );`
  - `glutIdleFunc( my_idle_func );`
  - `glutKeyboardFunc( my_key_events );`
  - `glutMouseFunc( my_mouse );`
Event Queue

MainLoop()

- Mouse_callback()
  - {}
  - ...
  - {}

- Keypress_callback()
  - {}
  - ...
  - {}

- window_callback()
  - {}
  - ...
  - {}

Event queue

Keyboard

Mouse

Window
Rendering Callback

- Callback function where all our drawing is done

- `glutDisplayFunc( my_display );`

```c
void my_display (void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glBegin( GL_TRIANGLES );
    glVertex3fv( v[0] );
    glVertex3fv( v[1] );
    glVertex3fv( v[2] );
    glEnd();
}
```
Idle Callback

- Use for animation and continuous update
- `glutIdleFunc(idle);`

```c
void idle( void )
{
    t += dt;
    glutPostRedisplay();
}
```
User Input Callbacks

- Process user input
- `glutKeyboardFunc( my_key_events );`

```c
void my_key_events (char key, int x, int y )
{
    switch ( key ) {
        case 'q': case 'Q':
            exit ( EXIT_SUCCESS);
            break;
        case 'r': case 'R':
            rotate = GL_TRUE;
            break;
    }
}
```
Mouse Callback

• Captures mouse press and release events

• `glutMouseFunc( my_mouse );`

```c
void myMouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        ...
    }
}
```
# Events in OpenGL

<table>
<thead>
<tr>
<th>Event</th>
<th>Example</th>
<th>OpenGL Callback Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keypress</td>
<td>KeyDown</td>
<td>glutKeyboardFunc</td>
</tr>
<tr>
<td></td>
<td>KeyUp</td>
<td></td>
</tr>
<tr>
<td>Mouse</td>
<td>leftButtonDown</td>
<td>glutMouseFunc</td>
</tr>
<tr>
<td></td>
<td>leftButtonUp</td>
<td></td>
</tr>
<tr>
<td>Motion</td>
<td>With mouse press</td>
<td>glutMotionFunc</td>
</tr>
<tr>
<td></td>
<td>Without</td>
<td>glutPassiveMotionFunc</td>
</tr>
<tr>
<td>Window</td>
<td>Moving</td>
<td>glutReshapeFunc</td>
</tr>
<tr>
<td></td>
<td>Resizing</td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>Idle</td>
<td>glutIdleFunc</td>
</tr>
<tr>
<td></td>
<td>Timer</td>
<td>glutTimerFunc</td>
</tr>
<tr>
<td>Software</td>
<td>What to draw</td>
<td>glutDisplayFunc</td>
</tr>
</tbody>
</table>
void RenderText(int x, int y, char *string) 
{
    int len, i;
    glRasterPos2f(x, y);
    len = (int) strlen(string);
    for (i = 0; i < len; i++)
        glutBitmapCharacter(
            GLUT_BITMAP_HELVETICA_18, string[i]);
}
OpenGL Basics

• Rendering
  – Converting geometric/mathematical object descriptions into frame buffer values

• OpenGL can render:
  – Geometric primitives
  – Bitmaps and Images (Raster primitives)
Graphics Pipeline

- OpenGL commands specify how each step is performed
- The graphics pipeline is modeled as a *state machine*
  - Each pipeline step has parameters (attributes) to be set
OpenGL as a Renderer

- Geometric primitives
  - points, lines and polygons
- Image Primitives
  - images and bitmaps
  - separate pipeline for images and geometry
    - linked through texture mapping
- Rendering depends on state
OpenGL – a state machine

• Once you put it in a state, it remains in this state until you explicitly change it
• Examples of state variables: current color, projection transformation, light properties, material properties etc.
OpenGL Command Format

```
glVertex3fv( v )
```

- **Number of components**
  - 2 - (x, y)
  - 3 - (x, y, z)
  - 4 - (x, y, z, w)

- **Data Type**
  - b - byte
  - ub - unsigned byte
  - s - short
  - us - unsigned short
  - i - int
  - ui - unsigned int
  - f - float
  - d - double

- **Vector**
  - omit "v" for scalar form
  - glVertex2f( x, y )
OpenGL Geometric Primitives

- The geometry is specified by vertices.
- There are ten primitive types:
Vertices and Primitives

- Primitives are specified using
  
  ```
  glBegin( primType );
  ...
  glEnd();
  ```

  - `primType` determines how vertices are combined

  ```
  GLfloat red, green, blue;
  GLfloat coords[3];
  glBegin( primType );
  for ( i = 0; i < nVerts; ++i ) {
    glColor3f( red, green, blue );
    glVertex3fv( coords );
  }
  glEnd();
  ```
An Example

```c
void drawParallelogram( GLfloat color[] )
{
    glBegin( GL_QUADS );
    glColor3fv( color );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( 1.0, 0.0 );
    glVertex2f( 1.5, 1.118 );
    glVertex2f( 0.5, 1.118 );
    glEnd();
}
```
Vertices and Primitives

- Points, **GL_POINTS**
  - individual points
Vertices and Primitives

• Lines, **GL_LINES**
  - pairs of vertices interpreted as individual line segments
Vertices and Primitives

- Line Strip, **GL_LINE_STRIP**
  - series of connected line segments
• Line Loop, **GL_LINE_LOOP**
  - Line strip with a segment added between last and first vertices
Vertices and Primitives

- Polygon, `GL_POLYGON`
  - boundary of a simple, convex polygon
Vertices and Primitives

- Triangles, **GL_TRIANGLES**
  - triples of vertices interpreted as triangles
Vertices and Primitives

- Triangle Strip, \text{GL\_TRIANGLE\_STRIP}
  - linked strip of triangles
Vertices and Primitives

- Triangle Fan,
  \textbf{GL\_TRIANGLE\_FAN}
  - linked fan of triangles
Vertices and Primitives

• Quads, \texttt{GL\_QUADS}
  - quadruples of vertices interpreted as four-sided polygons
Vertices and Primitives

- Quad Strip, `GL_QUAD_STRIP`
  - linked strip of quadrilaterals
Vertices and Primitives

- Vertices may be specified in 2D, 3D, or 4D.
- 2D coordinates are promoted to 3D by assigning a Z value of zero.
- 4D homogeneous coordinates are reduced to 3D by dividing $x$, $y$, and $z$ by the $w$ coordinate (if non-zero).
Vertices and Primitives

- Between glBegin/ glEnd, those opengl commands are allowed:
  - glVertex*() : set vertex coordinates
  - glColor*() : set current color
  - glIndex*() : set current color index
  - glNormal*() : set normal vector coordinates
  - glTexCoord*() : set texture coordinates
Vertices and Primitives

- \texttt{glMultiTexCoord\*()} : set texture coordinates for multitexturing
- \texttt{glEdgeFlag\*()} : control drawing of edges
- \texttt{glMaterial\*()} : set material properties
- \texttt{glArrayElement()} : Extract array element data
- \texttt{glCallList()}, \texttt{glCallLists()} : execute display list
GLUI

- GLUI is a GLUT-based C++ user interface library which provides controls such as
  - buttons,
  - checkboxes,
  - radio buttons,
  - spinners, etc.
- It is window-system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management.
GLUI Controls
Standalone GLUI windows
GLUI subwindows

Hello World!
References

• Mainly from the resources of Ceng477 previous semesters, their references also valid for this document

• New references:
  – www.cse.unr.edu/~bebis/CS480/Notes/opengl_part1.ps.gz