Computer Graphics Course 2005

Introduction to GLUT, GLU and OpenGL

Administrative Stuff

- Teaching Assistant: Rony Goldenthal
- Reception Hour: Wed. 18:00 – 19:00 Room 31 (Ross – 1)
- Questions:
  - E-mail: cg@cs
  - Newsgroups: local.course.cg

Exercises

- ~6 exercises, can be submitted in pairs **(except ex0)**
- Programming Language: C/C++
- Programming Guidelines – see homepage
- Exercises planned to be:
  - Fun
  - Creative
  - Educational

What is OpenGL

- OpenGL is a software interface to graphics hardware.
- Mainly used for interactive 3D graphics
- Consists about 250 commands Available both in software and hardware over different environments
- Specifications set by leading industry companies

GLU - OpenGL Utility Library

- Higher level library - wraps some of OpenGL's functions.
- Provides modelling features such as: basic geometric primitives, polygons tessellation, quadric surfaces and NURBS
- Helps setting view and projection matrices.

GLUT - OpenGL Utility Toolkit

- OS independent windowing toolkit for graphics purposes
- Used mainly for educational purposes - to learn OpenGL
- Simple event-driven kit !
- Easy to write small applications based on OpenGL

Recognizing Command's Source

- OpenGL commands use **gl** prefix
- GLU commands use **glu** previx
- GLUT commands use **glut** previx

GLUT Basics: Running GLUT

- **glutMainLoop**()
  - Starting point of GLUT
  - Windows are displayed
  - Event processing started
  - After calling it, no direct control over program flow
  - Do not start rendering to a window before calling it

GLUT Basics: Initialization

- **glutInit**(int *argc, char *argv[])
  - Initializes GLUT and processes command line arguments.
  - Should be called before any other GLUT routine.
- **glutInitDisplayMode**(unsigned int mode) –
  - Specifies the window display mode, for example:
    - GLUT_RGB - sets RGB color mode instead of indexed-color
    - GLUT_DOUBLE - sets double buffered window instead of single
    - GLUT_DEPTH - enables depth buffered window.

GLUT Basics: Event Handling

- Once GLUT detects an event it calls the appropriate – 'callback' function (CBF)
- glut***Func() is used to connect an event to a user defined CBF (by passing a pointer to the CBF)
- Event types: window, mouse, keyboard, timer

GLUT Basics: Initialization

- **glutInitWindowPosition**(int x, int y)
  - specifies the initial screen location for the upper-left corner of the GLUT window.
- **glutInitWindowSize**(int width, int height)
  - specifies the initial window dimensions.
- int **glutCreateWindow**(char *string)
  - Creates a window for OpenGL purposes.
  - Returns the window's id.
  - **Warning:** window will not appear before **glutMainLoop** is called.

GLUT Basics: Window Events

- **glutDisplayFunc**(void (*func)(void)) –
  - handles window display (rendering)
- **glutReshapeFunc**(void (*func)(int w, int h))
- handles changes in window size.

GLUT Basics: Keyboard and Mouse Events

- **glutKeyboardFunc**(void (*func)(unsigned char key, int x, int y))
  - handles keyboard strokes
- **glutMouseFunc**(void (*func)(int button, int state, int x, int y)) –
  - handles mouse buttons events – press/release
  - button = GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON
  - state = GLUT_DOWN, GLUT_UP
- **glutMotionFunc**(void (*func)(int x, int y)) –
  - handles mouse movement events (while one of the buttons is pressed - dragging)

GLUT Basics: Timer Event

- **glutTimerFunc**(int millis, void (*func)(int value), int value)
  - Called once in millis time (from now) and will send value as the argument.
- **glutIdleFunc**(void (*func)(void))
  - Called whenever the event loop is idle
  - Used to manage background tasks

GLUT Basics: Other Commands

- **glutSwapBuffers( )**
  - used in double buffer mode, in the display function
- **glutPostRedisplay()**
  - Notifies GLUT that the window needs to be redrawn
  - **Never** call the display function directly

OpenGL Command Syntax

- All OpenGL commands start with **gl**.
- Defined constants begin with GL_ and are all capital
  - Example: GL_COLOR_BUFFER_BIT
- Suffix tells us which data type the function accepts:
  - b – signed char: **GLbyte**
  - ub – unsigned char: **GLubyte**
  - i – 32 bit integer: Glint
  - f – 32 bit floating point: GLfloat
  - d – 64 bit floating point GLdouble
  - glVertex2**f**(GLfloat x, GLfloat y) vs. glVertex2**i**(GLint x, GLint y)

OpenGL Command Syntax

- A number in the suffix specifies number of parameters accepted:
- 'v' specifies that this variant accepts an array or pointer as parameter:
  - glVertex**2**i(GLint x, GLint y) vs. glVertex**3**i(GLint x, GLint y, GLint z)
  - glVertex4d**v**(GLdouble[4] vector) - one array of doubles of length of 4.

OpenGL as a State Machine

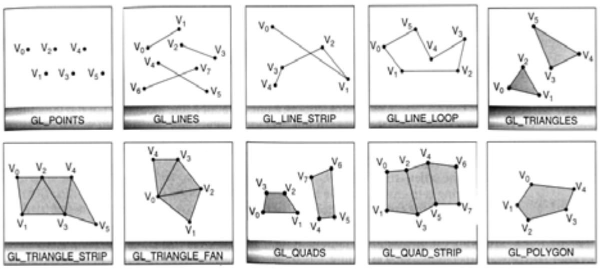
- OpenGL is a state machine, therefore many of its commands change inner states such as color and other drawing modes.
- **glClear**(<buffer_const>) - clears the buffer indicated by the const argument:
  - GL_COLOR_BUFFER_BIT - for color buffer(RGBA)
  - GL_DEPTH_BUFFER_BIT - for depth buffer
  - GL_ACCUM_BUFFER_BIT - for accumulation buffer
  - GL_STENCIL_BUFFER_BIT - for stencil buffer
- **glClearColor**(double red, double green, double blue, double alpha) -sets the clear color (0.0 - 1.0).
- **glClearDepth**(double depth) - sets the depth value.

OpenGL as a State Machine

- OpenGL is a state machine
- You put it in a certain state
  - Remains in effect until state is changed
- Example: glColor() sets current drawing color.
  - Once called all shapes will be drawn using this color
  - Until next call of glColor
- More states: current transformation, viewing and projection parameters, lighting parameters, line width
- Many states are either enabled or disabled.
  - glEnable()
  - glDisable()

OpenGL - Drawing Geometric Primitives

- **glBegin**(GLenum *mode*) sets the type of primitive OpenGL will interpret the next vertices list:



OpenGL - Drawing Geometric Primitives

- **glColor**{34}{b s i f d ub us ui}[v](*...*) sets drawing color (**in RGBA mode**). Colors are defined by a combination of Red, Green and Blue intensity components (and alpha channel).
- Examples:
  - glColor3f(1.0, 0.0, 0.0) ; defines Red color
  - glColor3f(0.5, 0.5, 0.5) ; defines Grey color
  - glColor3ub(0, 255, 0) ; defines Green color
  - glColor3dv(c) ; whereas c is - double c[3] ;
- Colors input range are type dependent (see OpenGL programming guide V1.2 page 168)

OpenGL - Drawing Geometric Primitives

- **glVertex**{234}{sifd}[v](*coords*) this command specifies a vertex, example:
  - glVertex2f(100.0,50.0) ;
  - glVertex3iv(vector) ; whereas v is int v[3].
- glVertex2XX sets the third coordinate to be 0 and the fourth to be 1.0, glVertex3XX sets the fourth coordinate to be 1.0

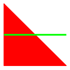
OpenGL - Drawing Geometric Primitives

- **glBegin**(GLenum *mode*)
  - Starts the vertex drawing mode
- **glEnd**() - Marks the end of vertex-data list.
- **glFlush**() Forces previously issued OpenGL commands to begin execution.
- **glFinish**() Forces all previously issued OpenGL commands to complete. This command doesn't return until all previous commands are fully realized.

OpenGL - Drawing Geometric Primitives

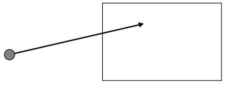
- Example code:
  - glClearColor(0.0, 0.0, 0.0, 0.0) ;
  - glClear(GL_COLOR_BUFFER_BIT) ;
  - glColor3f(1.0, 0.0, 0.0) ;   /* red color */
  - glBegin(GL_TRIANGLES) ;
  - glVertex2f(0.0, 0.0) ; glVertex2f(1.0, 0.0) ; glVertex2f(1.0, 1.0) ;
  - glEnd() ;
  - glColor3f(0.0, 1.0, 0.0) ;   /* green color */
  - glBegin(GL_LINES) ;
  - glVertex2f(0.0, 0.5) ; glVertex2f(1.0, 0.5) ;
  - glEnd() ;
  - glFlush() ;

Result:

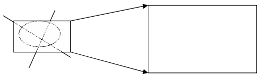


OpenGL - 2D Viewing Transformation

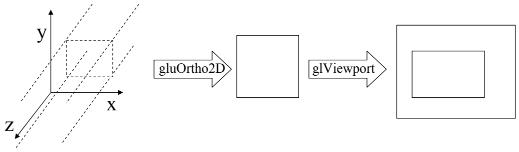
- 2D Coordinate System specification:
- Where will a given vertex be mapped on the screen?



- Thinking the question over we should be able to specify which rectangle in "vertices" coords. sys. will be mapped to the screen



OpenGL - 2D Viewing Transformation

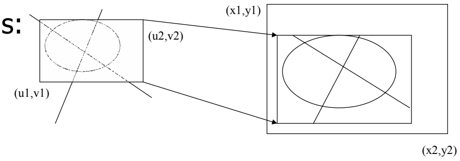


OpenGL - 2D Viewing Transformation

- This is done by the next four commands:
  - glViewport(u1, v1, u2, v2) ;
  - glMatrixMode(GL_PROJECTION) ;
  - glLoadIdentity() ;
  - gluOrtho2D(x1, x2, y1, y2) ;
- The above four lines maps the rectangle(x1, y1, x2, y2) in the "vertices" coords. sys. to the (u1, v1, u2, v2) in the window.

OpenGL - 2D Viewing Transformation

- That is:



- gluOrtho2D performs parallel projection of a rectangle in the "vertices" coordinate system to a **canonical square** in the interval -1,1. The axis of projection is the Z-axis (the third coord in glVertex)
- glViewport maps this **canonical square** to the given windows coordinates.