What is computer graphics?

- Computer graphics is concerned with Producing pictures using computer.
  - Creating pictures (modeling)
  - Presenting pictures (rendering)
  - Interacting with pictures

- The field of computer graphics
  - Methods and Principles (geometry, physics, optics, etc.)
  - Algorithms and data structures
  - Software/hardware tools
  - Applications
### 3D computer graphics paradigms:

- **Traditional (this course):** modeling → rendering
  - User interface
  - Modeling → Rendering

- **Other paradigms**
  - Data visualization
  - Image-base rendering

- **Computer graphics vs. image processing/vision**
  - Graphics
  - Models → pictures → Image processing/vision

### Computer graphics applications

- **Entertainment:** movie production, animation, special effects, games,
- **Presentation:** Web, e-commerce, publishing, computer art, advertisement, education.
- **Design:** CAD/CAM, ECAD, fashion design, architectural design.
- **Simulation:** complex experiments, flight simulation, surgical simulation, traffic control, power plants, etc.
- **Graphical user interface (GUI):** bitmap window systems, interface tools, VR systems.
- **Scientific visualization:** biological image analysis, medical diagnosis, weather data, satellite data, remote sensing, NASA data, simulation data, etc.
Display devices

✓ Line-drawing displays (vector display)
  - Fast, accurate line drawing (e.g. plotter).
  - Early technology (vector graphics)
  - Cannot draw smooth & shaded surfaces
✓ Raster display
  - Current technology (raster graphics)
  - Pixel based raster screen (CRT, LCD)
  - Supported by frame buffer
  - High quality realistic images
Typical raster graphics system

Graphics processor: a specialized processor for geometric and graphical display operations.
Frame buffer

- Pixel: the smallest screen unit that can be lighted with a color value independently (Fig. 1.35, Fig. 1.37)
- Raster: 2D array of pixels
- Frame buffer: a 2D array of memory cells made of high speed memory chips, corresponding one-to-one to the raster.
- Frame buffer cell: the frame buffer memory cell (often called pixel as well)
  - 1 bit: black and white images
  - 8 bits: 256 colors or color index
  - 24 bits: $2^{24}$ RGB colors
  - 32 bits: $2^{32}$ RGBA colors

Red (8 bits) | Green (8 bits) | Blue (8 bits)
RGB color scheme

✓ RGB color

| Red (8 bits) | Green (8 bits) | Blue (8 bits) |

✓ Indexed color (LUT)  (Fig. 1.40)
- Color lookup table: a programmable association between frame buffer color codes and the final displayed colors
- Palette: The number of different colors the system is capable of displaying (not all at the same time), or $2^n$, where $n$ is the LUT width.
CRT

- Three electron beams: R, G, B beams (Fig. 1.38)
- Electron beam refreshing rate: 50-60 frames per second.
- Flicker effect
- Interlaced refreshing: odd rows and even rows are displayed alternately at half of the normal rate.
- Non-interlaced refreshing: scanline by scanline, top to bottom, left to right, at normal refreshing rate.
LCD (Liquid-Crystal Display)

✓ A flat-panel display.
✓ Producing a picture by passing polarized light from a light source through a liquid-crystal material that can be aligned to either block or transmit the light.
✓ A color display is made possible by placing color filters over each display unit. Red, green and blue dots are obtained through the use of filters for each of the three primary colors.

LCD vs. CRT

✓ LCD advantages
  - No flickers
  - More compact (smaller and lighter)
  - More eye-friendly

✓ LCD disadvantages
  - More expensive
  - Fixed resolution
  - Color fidelity not as good as in CRT
Synthetic image formation

- **Object modeling:**
  - 3D geometric models
  - graphics primitives
  - material and optical properties
  - View independent

- **Lighting:**
  - Lights
  - ray tracing model

- **Synthetic camera model**

---

Synthetic camera model

- Center of Projection (COP)
- Viewing plane
- Object model
Graphics architecture

✓ **Graphics pipeline**

- Scene modeling
- Viewing transformation
- Clipping
- Projection
- Rasterization

✓ **Display list**
- list of object currently being displayed, residing in the graphics processor.
- The list may be updated once a command is executed that needs to change the display.

---

Graphics pipeline

✓ **Scene modeling**: geometric representation of shape information, material & optical properties, lighting condition

✓ **Viewing transformation**: transform objects from the scene coordinate system (world coordinate system) to camera coordinate system (viewing coordinate system).

✓ **Clipping**: determining the parts of the objects that are within the given viewing region.

✓ **Projection**: 3D objects to 2D objects on viewing plane

✓ **Rasterization**: generating pixels that display the projected objects.
Graphics API
(application programming Interface)

- A high level conceptual model of the graphics system, providing:
  - Object definitions
  - View specification
  - Lighting environment

- We will study both sides of the API: application programming, and the implementation of the API (graphics pipeline)

(Graphics Standards)

- Purpose
  - Avoid system-dependent details
  - Platform-independent applications
  - Common graphics languages

- 2D standard: X-window, GKS, postscript
- 3D standard: PHIGS, PEX, OpenGL

- Common features: object description, attributes, viewing functions, transformations, lighting, interactions, controls, etc.
OpenGL Overview

- Built on top of window system and OS

- Client-server model: OpenGL commands issues by client are processed by OpenGL server.

- Platform independence:
  - No windowing operations
  - Does not handle input devices (GLUT)

OpenGL libraries

- Application programming level

  - OpenGL: OpenGL library
  - GLU: OpenGL utility library
  - GLUT: Interface library

  - Xlib: Low-level system functions

Diagram:
- Application programs
- OpenGL
- GLU
- GLUT
- Xlib
- Low-level system functions
OpenGL & GLUT examples

- A simple rotating cube example

- More examples:

- Header files: gl.h, glu.h, glut.h, glaux.h
- Library files: opengl32.lib, glu32.lib, glaux.lib, glut32.lib, opengl32.dll, glu32.dll, glut32.dll.

OpenGL Syntax

- Prefix “gl” for OpenGL functions, “glu” for “GLU” functions, and “glut” for GLUT functions, e.g. 
  `glClearColor()`, `gluLookAt()`, `glutCreateWindow()`
- Prefix “GL_” for constants, e.g.
  `GL_COLOR_BUFFER_BIT`
- Suffix (e.g. “2f”, “3fv”) indicates the number and type of parameters, e.g. `glVertex3f()`, `glVertex3fv()`
OpenGL Data Types

<table>
<thead>
<tr>
<th>suffix</th>
<th>data type</th>
<th>Ansi-C type</th>
<th>OpenGL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>8 bit integer</td>
<td>signed char</td>
<td>GLbyte</td>
</tr>
<tr>
<td>s</td>
<td>16 bit integer</td>
<td>short</td>
<td>GLshort</td>
</tr>
<tr>
<td>i</td>
<td>32 bit integer</td>
<td>int, long</td>
<td>GLint, GLsizei</td>
</tr>
<tr>
<td>f</td>
<td>32 bit float</td>
<td>float</td>
<td>GLfloat</td>
</tr>
<tr>
<td>d</td>
<td>64 bit integer</td>
<td>double</td>
<td>GLdouble</td>
</tr>
<tr>
<td>ub</td>
<td>8 bit unsigned</td>
<td>unsigned char</td>
<td>Glubyte, GLboolean</td>
</tr>
<tr>
<td>us</td>
<td>16 bit unsigned</td>
<td>unsigned short</td>
<td>GLushort</td>
</tr>
<tr>
<td>ui</td>
<td>32 bit integer</td>
<td>Unsigned int</td>
<td>GLuint, GLenum</td>
</tr>
</tbody>
</table>

OpenGL as a state machine

- OpenGL state is defined by state variables
- State variables can be explicitly set (e.g. glColor) or enabled/disabled (for Boolean values) by glEnable or glDisable
- glPushAttrib and glPopAttrib: store and restore the system’s state
- State variables have default values
- Query state variable values: glGetBoolean, glGetDoublev, glGetFloatv, glGetIntegerv
Drawing OpenGL primitives

- Drawing mode: Immediate mode and Display list mode

- Colors
  - `glClearColor()`: background
  - `glClear()`: clear buffer
  - `glColor3f()`: set current color

- OpenGL primitives
  - `glVertex*`: defining vertices of primitives
  - Vertices are defined in "glBegin ..... glEnd" block (color and other attributes may also be defined in the block)

- `glFlush`: forcing completion of drawing
Attributes

✓ Attributes (as state variables): color, point size (glPointSize), line width (glLineWidth), text font, polygon filling mode, line pattern, etc.

✓ RGB Color
  - blending of red, green and blue components, defined by glColor**(). RGB color values are between 0.0 and 1.0
  - Alpha component: opacity value, used to composite transparent surfaces.

✓ Color index mode
  - store color index in frame buffer, and then use a color lookup table to generate RGB values.
  - glIndex

Example

```c
#define PI 3.14159265

GLint circle_points = 100;

 glBegin (GL_LINE_LOOP);
  glColor3f (1.0, 1.0, 0.0);
  For (int i=0; i < circle_points; i++) {
    angle = 2.0 * PI * i / circle_points;
    glVertex2f (cos(angle), sin(angle));
  }
 glEnd ();
```
OpenGL polygons

✓ Simple (non-selfintersecting) and convex polygons only (concave polygons are available in GLU)
✓ Polygon face
  - front face: counterclockwise
  - back face: clockwise
✓ Normals at the vertices: glNormal
✓ glPolygonMode (face, mode)
  - Face: GL_FRONT, GL_BACK or GL_FRONT_BACK
  - Model: GL_FILL or GL_LINE
✓ Shading model : glShadeModel (mode)
  - Mode: GL_FLAT or GL_SMOOTH

Control functions and GLUT

✓ System initialization:
  - glutInit (int *argcp, char **argv);
  - glutInitDisplayMode (GLUT_Double | GLUT_RGB);
    Other modes include: GLUT_SINGLE, GLUT_DEPTH, etc.
✓ Windows:
  - glutInitWindowPosition (x0, y0):
  - glutInitWindowSize (w, h);
  - glutCreateWindow (char *name);
  - xo, yo, w, h are in screen (pixel) coordinate system
✓ Main loop : glutMainLoop ( ):
  Put the system in an event-waiting state.
GLUT example

Void main () {
    glutInit ();
    glutInitDisplayMode (GLUT_RGB | GLUT_SINGLE);
    glutInitWindowSize (640, 480);
    glutInitWindowPosition (100, 150);
    glutCreateWindow ("my window");

    glutDisplayFunc (mydisplay);
    glutReshapeFunc (myreshape);
    glutMouseFunc (myMouse);
    glutKeyboardFunc (myKeyboard);
    glutMainLoop ();
}

// appropriate #includes go here - see Appendix 1
void main(int argc, char** argv) {
    glutInit( &argc, argv ); // initialize the toolkit
    glutInitDisplayMode( GLUT_SINGLE | GLUT_RGB ); // set the display mode
    glutInitWindowSize(640,480); // set window size
    glutInitWindowPosition(100,150); // set the window position on screen
    glutCreateWindow("my first attempt"); // open the screen window

    // register the callback functions
    glutDisplayFunc (mydisplay);
    glutReshapeFunc (myreshape);
    glutMouseFunc (myMouse);
    glutKeyboardFunc (myKeyboard);

    myInit(); // additional initializations as necessary
    glutMainLoop(); // go into a perpetual loop
}

void myInit(void) {
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
GLUT: display callback

✓ Display: glutDisplayFunc (void *func) ()
   function to be called when a display is required
✓ glutPostRedisplay (): explicit calling of display function
✓ Example

  Void myDisplay () {
    glClear (GL_COLOR_BUFFER_BIT);
    glBegin (GL_POLYGON);
    glVertex3f (100.0, 100.0, 0.0);
    glVertex3f (200.0, 100.0, 0.0);
    glVertex3f (200.0, 200.0, 0.0);
    glEnd;
    glFlush ();
  }

GLUT: reshape callback

✓ Reshape: glutReshapeFunc (void (*func) (int w, int h))
   function to be called when the window size is changed.
✓ Example

  Void myReshape (int width, int height) {
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D (0.0, width, 0.0, height);
    glMatrixMode (GL_MODELVIEW);
  }
Example: Sierpinski gasket

Definition:
- Start with a triangle $T_0$, $T_1$, $T_2$
- Let $P_0$ be one of $T_0$, $T_1$, $T_2$ at random
- For $k=1$ to $N$
  - Let $T$ be one of $T_0$, $T_1$, $T_2$ at random
  - $P_k = (T + P_{k-1}) / 2$
  - Draw $P_k$
void Sierpinski(void)
|
| GLintPoint T[3] = {{10,10},{300,300},{200,300}};
|
| int index = random(3); // 0, 1, or 2 equally likely
| GLintPoint point = T[index]; // initial point
| drawDot(point.x, point.y); // draw initial point
| for(int i = 0; i < 1000; i++) // draw 1000 dots
| |
| index = random(3);
| point.x = (point.x + T[index].x) / 2;
| point.y = (point.y + T[index].y) / 2;
| drawDot(point.x, point.y);
| |
| glFlush();
|

void drawDot(GLint x, GLint y)
|
| // draw dot at integer point (x, y)
| glBegin(GL_POINTS);
| glVertex2f(x, y);
| glEnd();
|
---

**More examples**

- **Dot plotting of functions:**
  - Fig 2.15, Fig 2.16

- **Polyline plotting:**
  - Fig 2.19, Fig 2.20, Fig 2.21

- **Parameterizing figures:**
  - Fig 2.24, Fig 2.25

- **Draw filled polygons**
\[ f(x) = e^{-x} \cos(2\pi x) \]
```c
#include <fstream.h>
void drawPolyLineFile(char *fileName)
{
    ifstream inStream;
    inStream.open(fileName, ios::in); // open the file
    if(inStream.fail())
        return;
    glClearColor(GL_COLOR_BUFFER_BIT); // clear the screen
    GLint numpolys, numlines, x, y;
    inStream >> numpolys; // read the number of polylines
    for(int j = 0; j < numpolys; j++) // read each polyline
    {
        inStream >> numlines;
        glBegin(GL_LINES_STRIP); // draw the next polyline
        for(int i = 0; i < numlines; i++)
        {
            inStream >> x >> y; // read the next x, y pair
            glVertex2i(x, y);
        }
        glEnd();
        glFlush();
    }
    inStream.close();
}
```
void parameterizedHouse(GLintPoint peak, GLint width, GLint height)
// the top of house is at the peak; the size of house is given
// by the height and width
{
    glBegin(GL_LINE_LOOP);
    glVertex2l(peak.x, peak.y); // draw shell of house
    glVertex2l(peak.x + width / 2, peak.y - 3 * height / 8);
    glVertex2l(peak.x + width / 2, peak.y - height);
    glVertex2l(peak.x - width / 2, peak.y - height);
    glVertex2l(peak.x - width / 2, peak.y - 3 * height / 8);
    glEnd();
    draw the chimney in the same fashion
    draw the door
    draw the window
}
Event-driven interface

- **Callback functions**: when an input device is triggered, a callback function associated with the event is called to carry appropriate action.

  - Application program:
    - callback-1()
    - callback-2()
    - ...
    - callback-n()

- **Common events**: display, reshape, mouse, keyboard, motion, idle.

More callback functions

- **Mouse callback**
  
  ```c
  Void myMouse (int btn, int state, int x, y) {
      If (btn == GLUT_LEFT_BUTTON
          && state == GLUT_DOWN) exit();
  }
  ```

- **Keyboard callback**
  
  ```c
  Void myKeyboard (unsigned char key, int x, int y) {
      If (key == 'q' || key == 'Q') exit();
  }
  ```
More callback functions

- Motion callbacks: mouse moving with some button down
  
  \[ \text{glutMotionFunc (void (*func) (int \ x, \ int \ y))} \]

- Passive motion callback: mouse moving without any button down
  
  \[ \text{glutPassiveMotionFunc (void (*func) (int \ x, \ int \ y))} \]

- Idle callback: called every main loop cycle
  
  \[ \text{glutIdleFunc (void (*func) (void))} \]

Examples

- Controlling the sierpinski gasket
  - Example 2.4.3

- Interactive polyline drawing
  - Example 2.4.4 (Fig 2.39, 2.40)

- Freehand drawing
  - Example 2.4.5

- Keyboard interaction
  - Fig 2.41
Controlling the sierpinski gasket

static GLintPoint corners[3];
static int numCorners = 0;
if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
    corners[numCorners].x = x;
    corners[numCorners].y = screenHeight - y;
    if (++numCorners == 3) {
        Sierpinski (corners);
        numCorners = 0;
    }
}

Freehand drawing

void myMovedMouse (int mouseX, int mouseY)
{
    GLint x = mouseX;
    GLint y = screenHeight - mouseY;
    GLint brushSize = 20;
    glutRecti (x, y, x+brushSize, y+brushSize);
    glutFlush ();
}
void myMouse(int button, int state, int x, int y)
{
    // Define NUM and initial values
    static GLint Point List[NUM];
    static int last = -1;
    // last index used so far

    // test for mouse button as well as for a full array
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && last < (NUM - 1))
    {
        List[++last].x = x;  // add new point to list
        List[last].y = screenHeight - y;  // get y from the bottom
        glClear(GL_COLOR_BUFFER_BIT);  // clear the screen
        glBegin(GL_LINE_STRIP);  // redraw the polyline
        for (int i = 0; i < last; ++i)
        {
            glVertex2f(List[i].x, List[i].y);
        }
        glEnd();
        glFlush();
    }
    else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    {
        last = -1;  // reset the list to empty
    }
}

void myKeyboard(unsigned char theKey, int mouseX, int mouseY)
{
    GLint x = mouseX;
    GLint y = screenHeight - mouseY;  // flip the y value as always

    switch (theKey)
    {
    case 'p':
        drawDot(x, y);  // draw a dot at the mouse position
        break;
    case GLUT_KEY_LEFT:
        List[++last].x = x;  // add a point
        List[last].y = y;
        break;
    case GLUT_KEY_RIGHT:
        exit(-1);  // terminate the program
        break;
    default:
        break;  // do nothing
    }
}
Menu and menu callback

- Definition (example)

```c
glutCreateMenu (mymenu);
glutAddMenuEntry ("quit", 1);
glutAddMenuEntry ("increase", 2);
glutAddMenuEntry ("decrease", 3);
glutAttachMenu (GLUT_RIGHT_BUTTON);

.................
void mymenu (int id) {
    if (id == 1) exit;
    else if (id == 2)  size = size * 2;
    else if (id == 3)  size = size / 2;
}
```

Submenu

```c
Sub_menu = glutCreateMenu (size_menu);
glutAddMenuEntry ("increase", 2);
glutAddMenuEntry ("decrease", 3);
glutCreateMenu (top_menu);
glutAddMenuEntry ("quit", 1);
glutAddSubMenu ("resize", sub_menu);
glutAttachMenu (GLUT_RIGHT_BUTTON);

...............  
void size_menu (int id) {
    if (id == 2) size *= 2; else if (id == 2) size /= 2;
}

void top_menu (int id) {  if (id == 1) exit;  }```
Some useful links

- **GLUT manual:**

- **OpenGL Reference Manual:**
  http://www.opengl.org/sdk/docs/
  http://www.glprogramming.com/blue/

- **OpenGL Programming Guide (redbook):**
  http://it-ebooks.info/book/2138/