Note 1. Introduction

I.1 General Introduction
- objectives
  * learn basic computer graphics terminology, concepts
  * design and implement simple 2D/3D interactive graphics related programs

- textbook:

- prerequisites
  * data structures (cs331)
  * matrices, vectors, analytic geometry (MATH230)
  * C/C++ programming

- reference material:
  * available from the web

- tutorials:
  * will be given regularly
I.2. What is Computer Graphics?

computer graphics: 3D model -> image
computational geometry: model <-> model
computer vision: image-> model
image processing: image <->image

- Computer Graphics: creation, storage and manipulation of models and images
- Interactive Computer Graphics: user controls contents, structure, and appearance of objects and their displayed images via rapid visual feedback.
- Start from 70’s, from drawing lines on CRT -> GKS -> OpenGL -> realistic graphics

---------- Interaction <-------- human ------------- ----------
    |                             |                   |
    |                             |                   |
    |                             |                   |
    |                             |                   |
    |                             |                   |
       (mouse, 3D devices)     (human)     (image)

(1) Modeling --------> (2) Rendering --------> (3) Displaying
(to describe)                          (to capture)                 (to show)
-> to the computer              -> the description             -> the image

Geometric Engine        Rendering Engine        Raster & Display Engine
Concerned with           Concerned with           Concerned with
- object geometry         - camera position         - hardware
- material property       - what is visible          - how to display
- lighting property       - color models            - illumination & shading
                        - raytracing & radiosity

** NOT COVERING THE FOLLOWING
- Adobe PhotoShop, AutoCAD, 3D studio Max, Digimation, Renderman
but the algorithms in the above (to some extent)

I.3. Modern Applications
- display of information: GIS, statistical plots, medicine, scientific visualization, CFD, climate models, geophysics, math, etc.
- design: CAD (architectural, mechanical, aircraft)
- simulation: flight simulator, 3D games, virtual reality.
- user interfaces: 3D better user interface.

Modern applications combine computer processing power, networks and graphics.

I.4 CRTs

Early calligraphic CRT:
Modern CRT: raster based:

- hitting phosphors (organic compound)
  - when exposed to energy, absorbs and retains energy
  - when energy is removed, it will re-emit the energy
- needs refreshing. Typical frequency: 60 - 85Hz
- For color: 3 electronic guns: R, G, B
  - combination of 3 phosphor is a pixel. Pixel arrangement: … or .:
  - R.G.B. phosphors have different excitation level and releases energy at different frequencies
  - switch different combination of guns on/off for other colors
  - intensity $\propto$ number of electrons hitting phosphor (control: heated cathode voltage and control grid voltage)
- refresh a pixel before the pixel fades out.

Terminology:
- non-interlace vs interlace

CS527 Computer Graphics - Introduction
• pixel: picture element
• dot pitch: shortest distance between two dots of the same color (in mm)
• resolution: maximum number of points that can be displayed without overlapped on a CRT (total), or, number of points per unit area that can be plotted horizontally and vertically (dpi)

I.5 Display Pipeline

I.6 Frame Buffer and Video Controller
Example 1: 8 color display

<table>
<thead>
<tr>
<th>R</th>
<th>G</th>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Red</td>
</tr>
</tbody>
</table>

CS527 Computer Graphics - Introduction
Eg. 2 24 bit display

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Blue</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Yellow</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Cyan</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Magenta</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>White</td>
</tr>
</tbody>
</table>

Memory requirement: 1280 x 1024 x 3 bytes (24 bits) ≈ 4 Mb

Eg. 3 Color Index – lookup table

Eg. N=8, W=4
Effectively 12 bits color which is 2^12 = 4096 colors
But only 2^8 = 256 available at any one time
II. Graphics Architecture

Now, the question is how to draw in the frame buffer.
2D: images, line segments, polygons, and other 2D geometric objects.
3D?

To simulate how it works in optical systems and human eyes.

- 3D objects: position of geometric primitives, in most cases, vertices linked together.
- Viewer
- Light source: point source – ideal. Light spectrum & directions
- Ray tracing. Reflect, diffuse, pass through.
  An image-formation technique, complex, computing intensive.
  To make it simpler: 1. point light source; 2. no difference whether an object reflects lights or
  emits lights.

Camera model:
Principle:
  1. Specification of objects is independent of the specification of viewer. (An API should
     contain both.
  2. Images are computed using simple trigonometric calculations.
  3. Clipping.

Graphics architecture:
Closely related to the advances in computer hardware.
Pipeline architecture: Think about an automobile assembly factory.

How to partition the job:
Input: set of vertices defining primitives
Output: raster image

Vertices $\rightarrow$ Transformation $\rightarrow$ Clipper $\rightarrow$ Projection $\rightarrow$ Rasterization $\rightarrow$ Pixels
- Transformation: 3D geometry $\rightarrow$ geometry in the coordinate system of display
- Clipping: Ignore those parts that are outside of your view.
- Projection: 3D $\rightarrow$ 2D
- Rasterization: 2D objects $\rightarrow$ pixels

APIs:
Applications $\leftrightarrow$ Graphics Library $\rightarrow$ …
* objects; * viewers, * light sources; * material properties.

What could be automated???
III. 1. Graphics Standards and APIs

What do we need to put some graphics on the monitor?
- get a window
  - win32 API
  - xmotif
  - GLUT (GL Utility Toolkit)
- draw something on it
  - GKS (~KGB)
  - PHIGS, PHIGS+
  - OpenGL (successor of GL)
(demo: distort.exe)

Computer Graphics Software Standards – History:

- **Graphics Kernel System**
The first international standard for programming computer graphics applications.
  ➤ GKS is realized in the form of subroutine packages as a set of functions callable from a programming language.
  ➤ It supports the grouping of logically related primitives (such as lines, polygons, char strings) and their attributes into a collection called *segment* that **may not be nested**.
  ➤ It provides functions for picture generation, picture presentation, segmentation, transformation and input. It supports object hierarchy with transformations.

- **Programmer’s Hierarchical Interactive Graphics System**
  ➤ A *Structure* in PHIGS is a sequence of *elements* - primitives, appearance attributes, transformation matrices, and invocations of subordinate structures - whose purpose is to define a coherent geometric object.
  ➤ PHIGS effectively stores a special-purpose modeling hierarchy, complete with modeling transformations and other attributes passed as parameters to subordinate structures.
  ➤ Support nested hierarchical groupings of structures.

Non-official industry standards:
  ➤ SGI’s GL, OpenGL, 1993
  ➤ Microsoft’s Direct3D.

<table>
<thead>
<tr>
<th>VRML</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Java3D</td>
<td></td>
</tr>
<tr>
<td>OpenGL</td>
<td>Direct3D</td>
</tr>
</tbody>
</table>
II. 2. GLUT (GL Utility Toolkit)
- library addresses the problems of interfacing with window system
- provides the minimum functionality that should be expected in any modern windowing system
- Why choose GLUT: 1. simple, a minimal set of libs.
  2. available on both X and MSWindows
- Answer questions such as:
  * where on screen does the image appear?
  * how large will the image be?
  * how to create a window for an image?
  * how long will the image remain?

Sample code: will be needed for all our lab assignments.

```c
#include <GL/glut.h>
void main(int argc, char **argv)
{
    glutInit(&argc, argv); /* initiate the interaction, pass args to X */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0,0); /* allocate frame buffer */
    glutCreateWindow("simple OpenGL example"); /* create the window */
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

For interaction: glutMouseFunc(); glutKeyboardFunc();

`glutInit()` initializes the GLUT library and negotiates a session with the window system. During the process, `glutInit` may cause the termination of the GLUT program with an error message if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options. `glutInit` also processes command line options, but the specific options are window system dependent.

`glutInitDisplayMode()` is used when creating top-level windows, subwindows, and overlays to determine the OpenGL display mode. GLUT_SINGLE mask to select a single buffered window. This is the default if neither GLUT_DOUBLE or GLUT_SINGLE are specified. GLUT_RGB selects an RGBA mode window.
The intent of the initial window position and size values is to provide a suggestion to window system for a window’s initial size and position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified size or position. A GLUT program should use the window’s reshape callback to determine the true size of the window.

`glutCreateWindow()` creates a top-level window. The name will be provided to the window system as the window’s name. The intent is that the window system will label the window with the name.

`glutDisplayFunc()` sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. GLUT determines when the display callback should be triggered based on the window’s redisplay state. The redisplay state for a window can be either set explicitly by calling `glutPostRedisplay()` or implicitly as the result of window damage reported by the window system.

`glutMainLoop` enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

Try the Sierpinski Gasket program.
II. 3. OpenGL

- OpenGL provides a powerful but primitive set of rendering commands (functions) to produce high-quality color images of 2D or 3D objects. All higher-level drawing must be done in terms of these commands.
- It is developed by SGI as a hardware independent API for use across all graphics workstations equipped with this library.
- The basic part contains about 120 functions.
- Additional libraries are needed to allow programmers to work easily with the windowing system and the I/O subsystem. These libraries include:
  - GLU (OpenGL Utility Library)
  - GLUT (OpenGL Utility ToolKit)
  - GLX (OpenGL Extension to the X window System)
  - AUX (OpenGL Auxiliary Library)

- Library organization for X windows

void myinit(void)
{
  /* attributes */
  glClearColor(1.0, 1.0, 1.0, 1.0); /* white background */
  glColor3f(1.0, 0.0, 0.0); /* draw in red */
  glPointSize(2.0); /* render points at the size of 2 pixels. */

  /* set up viewing */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0, 500.0, 0.0, 500.0);
  glMatrixMode(GL_MODELVIEW);
}

CS527 Computer Graphics - Introduction
} void display(void)
{
    typedef GLfloat point2[2]; /* define a point data type */
    point2 vertices[3] = { {0.0, 0.0}, {250.0, 500.0}, {500.0, 0.0} };

    int I, j, k;
    long random();
    point2 p = {75.0, 50.0}; /* just a point to start */

    glClear(GL_COLOR_BUFFER_BIT); /* clear the window */

    /* computes and plots 5000 new points */
    for (k = 0; k < 5000; k++)
    {
        j = random()%3 /* pick a vertex at random */

        /* compute point halfway between vertex and old point */
        p[0] = (p[0] + vertices[j][0]) / 2.0;
        p[1] = (p[1] + vertices[j][1]) / 2.0;

        /* plot point */
        glBegin(GL_POINTS);
        glVertex2fv(p);
        glEnd();
    }

    glFlush();
}

(a) **state machine**: OpenGL is designed as a state machine. It maintains a set of records that defines its current state. A record may specifies:
(i) line attribute: color, style or width
(ii) polygon attribute: draw as points, outlined, filled, or patterned
(iii) color attribute: current color to be shared by all attributes
(iv) et.c
Once an attribute is set, it remains until the next change

(b) **Matrix modes**
- pipeline graphics systems have an architecture that depends on multiplying together, or concatenating, a number of transformation matrices to achieve the desired image of a primitive.
values of these matrices are part of the state of the system and remain in effect until changed.
- Two important matrices are the model-view and projection matrices.
- matrix operations can be applied to any type of matrix; thus, we must first set the matrix mode. Default: model-view matrix.

(c) Views / Projections
- viewing volume: a volume where we see the items

- orthographic projection vs perspective projection

(d) OpenGL primitives:

glVertex{2|3|4}{i|s|d|f}[v] (p);

glBegin(Glenum mode); mode can be: GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP, GL_POLYGON.
glEnd();
glFlush();

True colors: glColor3f(r,g,b); glClearColor(,,,);
Index colors: glIndexi(i); - select color
  Setting color involves window management:
glutSetColor(i, r, g, b);
Orthographic view: glOrtho(left, right, bottom, up, near, far);

Matrix mode: select from Model-view and projection.

(e) Other primitive: self discovery....

Video Demo: Bingo—a short movie created by Chris Landreth with Maya software of Alias|Wavefront Inc., which is now a subsidiary of Silicon Graphics. This movie has been selected as the grand finale of the Electronic Theatre at SIGGRAPH’98.
III. Input and Interaction

OpenGL does NOT support interaction!!!

1. Input Devices

Point Devices: 2D: Mouse, Trackball, Light Pen, Data Tablet, Touch Screen, Joystick
   3D: 3D wand, data glove
Logical Devices: String, Locator, Pick, Choice, Dial, Stroke

2. Input Mode

Request Mode:

Sample Mode:

Event Mode:

Callback functions

Client Server:
Which is a client and which is a server?
A server is where your graphics are rendered.

3. Display List

immediate mode; retained mode; why glFlush()?
glNewList(BOX, GL_COMPILE);
      
      glEndList();

      glCallList(BOX);

4. State Stack
Change state in a display list? A problem. The answer is state stack.

      glPushAttrib(GL_ALL_ATTRIB_BITS);
      glPushMatrix();
      
      glPopMatrix();
      glPopAttrib();

      Example: Display font
      Stroke font: display font use OpenGL
      Bitmap font: bitmap → Frame buffer

      glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN, int character);
      glutBitmapCharacter(GLUT_BITMAP_8_BY_13, int character);

5. GLUT Callback Functions

      int main()
      {
          glutInit();
          
          glutReshapeFunc(myReshape);
          glutMouseFunc(mouse);
          glutKeyboardFunc(keyboard);
          glutDisplayFunc(display);
          glutIdleFunc(idle);
          glutMainLoop();
      }

      void mouse(int button, int state, int x, int y)
      {
          if(button==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
              exit();
      }
Windows Event:
1. redraw?
2. aspect ratio?
3. scale?

void myReshape(GLsizei w, GLsizei h)
{
}

void idle();

Other callback functions: discover yourself.

Menues:

glutCreateMenu(mymenu);
glutAddMenuEntry("quit",1);
……
glutPostRedisplay();

void mymenu(int id)
{
    if (id==1) exit(0);
}

6. Animation

void display()
{
    glClear();
glBegin(GL_POLYGON);
    thetar = thetar * 2 * 3.1415926 /360;
    glVertex2f(…);
    ……
glEnd();


```c
void idle()
{
    thetha += 2;
    if (thetha >= 360) thetha -= 360;
    glutPostRedisplay();
}
```

Double Buffering:

If display is complex and cannot be drawn in a single refresh circle,
`glutSwapBuffers();`

Double buffering does not speed up drawing, it just ensures that you never see how the draw works.