A Short History

1963: Sutherland (MIT)
   Sketchpad
   Calligraphic display devices
   Interactive techniques
   Douglas Englebart invents the mouse.

1968: Evans & Sutherland founded

1969: First SIGGRAPH

Late 60’s to late 70’s: Utah Dynasty

1970: Pierre Bézier develops Bézier curves

1971: Gouraud Shading

1972: Pong developed

1974: Ed Catmull develops z-buffer (Utah)

1975: Bui-Young Phong

Mid 70’s: Raster Graphics (Xerox PARC, Shoup)
   Martin Newell’s team

1976: Jim Blinn develops image and texture mapping

1979: Turner Whitted introduces Ray Tracing

Mid 70’s - present: Quest for realism
   Radiosity; also mainstream real-time applications.

90’s: interactive environments, scientific and medical visualization, artistic
   rendering, image based rendering, etc.
History of Computer Animation

**Early 60’s:** Computer animations for physical simulation,
Edward Zajac displays satellite research using CG in 1961

**1973:** Westworld, The first film to use computer animation

**1974:** First Computer Animated Short, Hunger. Keyframe animation and morphing were used

**1977:** Star Wars, CG used for Death Star plans

**1982:** Tron, Wrath of Kahn. Particle systems and obvious CG

**1984:** The Last Star Fighter, CG replaces physical models Early attempt at realism using CG

**1986:** First CG animation nominated for an Academy Award: Luxo Jr.

**1989:** Tin Toy wins Academy Award

**1995:** Toy Story (1995, Pixar, Disney), the first full length fully computer-generated 3D animation
Rendering

The Graphics Rendering Pipeline

- **Rendering** is the conversion of a scene into an image:

- Scenes are composed of models in three-dimensional space. Models are composed of primitives supported by the rendering system.
- Models entered by hand or created by a program.
- The image is drawn on monitor, printed on laser printer, or written to a raster in memory or a file. Requires us to consider device independence.
- Classically, “model” to “scene” to “image” conversion broken into finer steps, called the graphics pipeline.
Overview of lecture 1

What is Computer Graphics?
generate images, (real time) animations

- **Computer Graphics: description → image**
- Image Processing: “bad” image → “better” image
- Pattern Recognition: image → description

CAD Examples
Color-coded wireframe display of body designs

Application Areas
- Computer Aided Design
- Presentation Graphics
- Computer Art
- Entertainment
- Education and Training
- Visualization
- Image Processing
- Graphical User Interfaces

CAD Examples
operating a tractor in virtual reality

headset view
Overview of lecture 1

Presentation Graphics Examples
- bar chart
- 2D 3D
- pie chart

Paintbrush Program Examples
- a Van Gogh look-alike
- an electronic watercolor
- both created with a cordless, pressure-sensitive stylus

Presentation Graphics Examples
- surface chart
- height field

Computer Art: Morphing
Overview of lecture 1

Education and Training Examples

- Color coded diagram used to explain the operation of a nuclear reactor

Education and Training Examples

- Flight simulator with 6 degrees of freedom in its motion

Education and Training Examples

- Automobile simulator to test driver reaction

Visualization of Molecules

- Visualization of protein structure
Overview of lecture 1

Visualization of Medical Data
- travelling in a colon

Image Processing Example
- land classification from satellite image:
  - original image - region boundaries - classification

Video Display Devices (1)
- cathode-ray tube (CRT)
  - refresh CRT (raster-scan display, random-scan display)
  - direct-view storage tubes
- refresh CRT
  - electrons hit phosphor-coated screen
  - permanent glowing through refreshing
  - phosphor persistence
  - resolution, e.g., 1280x1024, aspect ratio
  - screen diagonal size, e.g., 21 inches

Video Display Devices (2)
- raster-scan displays
  - television technology
  - object represented as set of discrete points
  - screen point: pixel or pel
- refresh buffer, frame buffer
  - bitmap or pixmap
  - 3Mbytes for 1024x1024x24
  - refresh rate (60 - 120 Hz)
  - horizontal, vertical retrace
- interlacing
Overview of lecture 1

Interrupt Scan Display Principle

RGB Shadow Mask Technology

Interactive Walkthrough

Input Devices (1)
- keyboard
- mouse (relative positioning)
- trackball (absolute positioning)
- spaceball (6 DOF)
- joysticks (stick for cursor steering)
- data glove (grasp virtual objects)
Graphics programming

Device independent programming and OpenGL: collection of routines that the programmer can call, along with a model of how the routines work together to produce graphics. The programmer sees only the interface and is shielded from the specific hardware and software of the resident graphics system.

Windows-based programming: managing the display of multiple overlapping windows. These can be moved and resized.

Event-driven programming: the program responds to various events. The system maintains event queue. The programmer organizes a program as collection of callback functions, executed when the event occurs. A callback function is created for every event that might occur. OpenGL comes with GLUT which assist in the event management.
Event driven programming

```c
main()
{
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);
}
```

Opening a window for drawing

```c
main()
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
}
```

Establishing coordinate system

```c
void myInit(void){
    glClearColor(0.0,1.0,1.0,0.0);
    glColor3f(1.0,0.0f,1.0f);
    glPointSize(4.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,440.0,0.0,480.0);}
```
void myInit(void)
{
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glColor3f(1.0, 0.0f, 1.0f);
    glPointSize(9.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 440.0, 0.0, 480.0);
}

void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
        glVertex2i(100, 50);
        glVertex2i(100, 130);
        glVertex2i(150, 130);
        glVertex2i(230, 100);
    glEnd();
    glFlush();
}

void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glutDisplayFunc(myDisplay);
    //glutReshapeFunc(myReshape);
    //glutMouseFunc(myMouse);
    //glutKeyboardFunc(myKeyboard);
    myInit();
    glutMainLoop();
}
First program - putting it together

Vertices - instead of points we work with vertex and create objects from vertices.

```c
glBegin(GL_POINTS);
    glVertex2i(100,50);
    glVertex2i(100,130);
    glVertex2i(150, 130);
    glVertex2i(230, 100);
    glVertex2i(230, 100);
glEnd();
```
Graphic functions:
- primitive (points, lines, polygons, pixels, text, curves, surfaces)
- attribute (colors, fills, type face for titles of graphs)
- viewing (types of views)
- transformation (rotation, translation, scaling)
- input
- control (communicate with window system, initialize programs, deal with errors)
Primitives and attributes

```
glBegin(GL_POINTS);
    .......

    glEnd();
```

### Points

- \( p_0 \)
- \( p_1 \)
- \( p_2 \)
- \( p_3 \)
- \( p_4 \)
- \( p_5 \)
- \( p_6 \)
- \( p_7 \)

### Lines

- \( p_0 \rightarrow p_1 \)
- \( p_1 \rightarrow p_2 \)
- \( p_2 \rightarrow p_3 \)
- \( p_3 \rightarrow p_4 \)
- \( p_4 \rightarrow p_5 \)
- \( p_5 \rightarrow p_6 \)
- \( p_6 \rightarrow p_7 \)
- \( p_7 \rightarrow p_0 \)

### Polyline

- \( p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_4 \rightarrow p_5 \rightarrow p_6 \rightarrow p_7 \rightarrow p_0 \)

### House

- \( p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_4 \)
class GLintPoint{
public:
GLint x, y;
};
void myInit(void)
{
}
GLintPoint CP;
void moveto(GLint x, GLint y)
{CP.x = x; CP.y = y; // update the CP}
void lineto(GLint x, GLint y)
{glBegin(GL_LINES); // draw the line
  glVertex2i(CP.x, CP.y);
  glVertex2i(x, y);
  glEnd();
  glFlush();
CP.x = x+10; CP.y = y+10; // update the CP}
void myDisplay(void)
{
  GLintPoint temp={100,200};
  GLint x[100], y[100];
  x[0]=30;
  y[0]=30;
  moveto(x[0],y[0]);
  for(int i = 1; i < 10; i++)
  {
    x[i]=x[i-1]+60;
    y[i]=y[i-1]+90;
    lineto(x[i], y[i]);
  }
  glFlush();
}
void main(int argc, char **argv)
{
  }
}
What will change with the red line change?
void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0,1.0,1.0,0.0); // white background
    glColor3f(0.6,0.6,0.6); // bright gray
    glRecti(20,20,100,70);
    glColor3f(0.2,0.2,0.2); // dark gray
    glRecti(70, 50, 150, 130);
    glFlush();
}

• by the opposite corners
• by the center point, height and width
• by the upper left corner, width and aspect ratio

aspect ratio = \frac{width}{height}
Polygons - object with border and interior

Polygon can be simple, convex, and flat.

Convex polygon:
a polygon is convex if a line connecting any two points of the polygon lies entirely within the polygon.
Polygon types

GL_POINTS

GL_POLYGON

GL_QUADS

GL_TRIANGLES

GL_TRIANGLES

GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN

GL_QUADS

GL_QUAD_STRIP
Interaction with mouse and keyboard

`glutMouseFunc (myMouse)` which registers `myMouse()` with the event that occurs when the mouse button is pressed or released;

`glutMotionFunc (myMovedMouse)` which registers `myMovedMouse()` with the event that occurs when the mouse is moved while one of the buttons is pressed;

`glutKeyboardFunc (myKeyboard)` which registers `myKeyBoard()` with the event that occurs when a keyboard key is pressed.
void drawDot(GLint x, GLint y) {
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

void myMouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) drawDot(x, screenHeight - y);
    else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) exit(-1);
}

void myDisplay(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
Draw rectangle at pressing of mouse button

Create polyline using the mouse

Freehand drawing with a fat brush

Interaction with keyboard
void myMouse(int button, int state, int x, int y)
{
    static GLintPoint corner[2];
    static int numCorners = 0; // initial value is 0
    if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        corner[numCorners].x = x;
        corner[numCorners].y = screenHeight - y; // flip y coordinate
        numCorners++; // have another point
        if(numCorners == 2)
        {
            glRecti(corner[0].x, corner[0].y, corner[1].x, corner[1].y);
            numCorners = 0; // back to 0 corners
        }
    }
    else if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    {
        glClear(GL_COLOR_BUFFER_BIT); // clear the window
        glFlush();
    }
}
void myMouse(int button, int state, int x, int y)
{
#define NUM 20
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; // window height is 480
    glClear(GL_COLOR_BUFFER_BIT); // clear the screen
    glBegin(GL_LINE_STRIP); // redraw the polyline
    for(int i = 0; i <= last; i++)
      glVertex2i(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 myMovedMouse(int mouseX, int mouseY)
{
  GLint x = mouseX; // grab the mouse position
  GLint y = screenHeight - mouseY; // flip it as usual
  GLint brushSize = 20;
  glRecti(x, y, x + brushSize, y + brushSize);
  glFlush();
}

glutMotionFunc(myMovedMouse);
void myKeyboard(unsigned char theKey, int mouseX, int mouseY) {
    int last;
    GLintPoint List[30];
    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 'E':
    case 'E':
        exit(-1); // terminate the program
        break;
    default:
        break; // do nothing
    }
Case study - pseudorandom cloud of dots

Text

Computer Graphics

Stroke text - ps font

Raster text
Curved objects - tessellation

Attributes

(a)  (b)

Color

![Graph showing color values over wavelength range 350 to 780 nm]
Frame buffer

Color lookup table

Red

Color lookup table

Green

Color lookup table

Blue

(a)

(b)
Clipping window

Viewport

Graphics window
What to expect in lecture 3?

Input and interaction - input devices, event-driven input, menu, animating

More drawing tools - clipping, relative drawing, drawing of circles