A SHORT HISTORY

1963: Sutherland (MIT)
   Sketchpad
   Calligraphic display devices
   Interactive techniques
   Douglas Engelbart invents the mouse.
1968: Evans & Sutherland founded
1969: First SIGGRAPH

Late 60’s to late 70’s: Utah Dynasty
1974: Pierre Bezier develops Bezier curves
1971: Computer Shading
1972: Pong developed
1974: Ed Catmull develops a buffer (Utah)
1975: Rui-Tsong Yang, Mid ’70’s: Raster Graphics (Xerox, PARC, Sun)

Martin Newell’s work 1976: Involved in image and texture mapping
1979: Turner Whitted introduces Ray Tracing
1970s – present: Quest for realism
   Radiosity; also mainstream real-time applications

90s: Interactive environments, scientific and medical visualization; artists
   rendering; image-based rendering, etc.

History of Computer Animation

Early 80’s: Computer animations for physical simulation.
   Edmond Borge, displays satellite research using CG in 1981
1973: Westworld, The first film to use computer animation
1974: First Computer Animated Short, Hunger, Keyframe animation and
   rendering were used
1977: Star Wars, CG used for Death Star plans
1982: Star Wars, The Empire Strikes Back, Particle systems and obvious CG
1984: The Last Starfighter, CG replaces physical models, Early attempt at
   realism using CG
1985: First CG animation nominated for an Academy Award: Limbo Jr.
1989: The Toy wins Academy Award
1989: Toy Story (1995, Pixar, Disney), the first full-length fully
   computer-generated 3D animation

Rendering

THE GRAPHIC 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 "raw" to "image" conversion broken into four
  steps, called the graphics pipeline.

CIS 470 Introduction to Computer Graphics/CSI - Dr. N. Georgieva
Overview of lecture 1

- What is Computer Graphics?
  - Generates images, real-time animation
- Application Areas
  - Computer Aided Design
  - Presentation Graphics
  - Computer Art
  - Entertainment
  - Education and Training
  - Visualization
  - Image Processing
  - Graphical User Interfaces

- CAD Examples
  - Color coded wireframe display of body designs
    - Aircraft
    - Automobile

- Presentation Graphics Examples
  - 2D 3D content

- Panorama Program Examples
  - Van Gogh look-alike
  - Electronic watercolor

- Computer Art: Morphing
  - Image transformation

- Education and Training Examples
  - Color coded diagrams used to teach the operation of a nuclear reactor
  - Flight simulator with 5 degrees of freedom in its motion

- Visualization of Molecules
  - Protein structure visualization

Overview of lecture 1

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
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);
}
```

### Establishing coordinate system

```c
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);
}
```

### 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");
}
```

### First program - putting it together

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

```c
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);
}
```

```c
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();
}
```

```c
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();
}
```
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

```cpp
class GLintPoint{
    GLint x, y;
public:
    GLintPoint(GLint x, GLint y) {this->x = x; this->y = y;}
    void myInit(void){}
    GLint x, y;
    void moveto(GLint x, GLint y){this->x = x; this->y = y; // update the CP}
    void lineto(GLint x, GLint y){glBegin(GL_LINES); glVertex2i(x, y); glVertex2i(this->x, this->y); glEnd(); this->x = x+10; this->y = y+10; // update the CP}
    void myDisplay(void){}
    void main(int argc, char **argv){
        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();
    }
};
```

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

\[ \text{aspect ratio} = \frac{\text{width}}{\text{height}} \]

Polygons - object with border and interior

Polygons 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.

![Convex Polygon Example](image)

Polygon types

![Polygon Types Diagram](image)
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.

```c
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
        glutPostRedisplay();
    }
}

void myMouse(int button, int state, int x, int y)
{
#define NUM 20
    static GLintPoint List[NUM];
    static int last = -1; // last index used so far
    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
        glBegin(GL_LINE_STRIP); // redraw the polyline
        glEnd();
        glutPostRedisplay();
    }
    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);
    glutPostRedisplay();
}

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':
        exit(-1); // terminate the program
        break;
    default:
        break; // do nothing
    }
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