

2. OpenGL - I

2.1 What is OpenGL?

- **Device-independent, application program interface (API)** to graphics hardware
- 3D-oriented
- Event-driven

Things OpenGL can do:

- wireframe models
- *depth-cuing* effect (lines farther from the eye are dimmer)
- anti-aliased lines
- flat-shaded polygons
- smooth-shaded polygons
- shadows and textures
- motion-blurred objects
- close-up shot
- atmospheric effect (fog)
- depth-of-the-field effect

2.2 Basic Structure of OpenGL Programs

Initialization Procedures

Callback Functions

```
void main ()
```

```
{
```

Window and coordinate system creation

State Initialization

Callback functions registration

Infinite Event Handling Loop

```
}
```

Classical (X Windows based) event handling approach:

```
void main () {  
    ...  
    ...  
    while (1) {  
        XNextEvent ( display, &event );  
        switch ( event.type ) {  
            case KeyPress:  
                { event handler for a keyboard event }  
                break;  
            case ButtonPress:  
                { event handler for a mouse event }  
                break;  
            case Expose:  
                { event handler for an expose event }  
                break;  
            ...  
            ...  
        }  
    }  
}
```

2.3 An OpenGL Example:

```
/*
 * This program demonstrates a dotdrawing process. Three dots
 * are drawn. Click the right button of the mouse to exit.
 */
#include <X11/Xlib.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

void myInit(void) {
    glClearColor (1.0, 1.0, 1.0, 0.0); // set black background color
    glColor3f (0.0f, 0.0f, 0.0f);    // set the drawing color
    glPointSize (4.0);                // set dot size 4 x 4
    glMatrixMode (GL_PROJECTION);    // set "camera shape"
    glLoadIdentity ();               // clear the matrix
    gluOrtho2D (0.0, 640.0, 0.0, 480.0); // set the World Window
}

void myDisplay(void) {
    glClear (GL_COLOR_BUFFER_BIT); // clear the screen
    glBegin(GL_POINTS);
        glVertex2i (100, 50);        // draw three points
        glVertex2i (100, 130);

```

```
    glVertex2i (150, 130);
glEnd();
glFlush ();          // send all out to display
}

void myMouse(int button, int state, int x, int y) {
    switch (button) {
        case GLUT_RIGHT_BUTTON:
            if (state == GLUT_DOWN) exit (-1);
            break;
        default:
            break;
    }
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);          // initialize the toolkit
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // set display mode
    glutInitWindowSize (640, 480); // set screen window size
    glutInitWindowPosition (100, 150); // set window position on screen
    glutCreateWindow (argv[0]);     // open the screen window
    myInit ();
    glutDisplayFunc(myDisplay);     // register redraw function
    glutMouseFunc(myMouse);        // register the mouse action function
    glutMainLoop();                // go into a perpetual loop
    return 0;
}
```

2.4 Include Files

Related libraries:

OpenGL

- include file: <GL/gl.h>
- GL routines use the prefix: *gl*

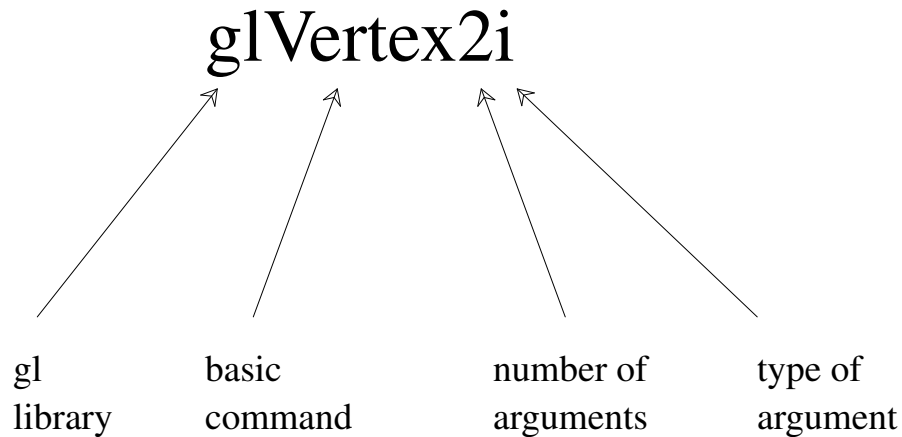
OpenGL Utility Library

- setting up matrices for viewing transformation
- performing polygon tessellation
- rendering surfaces
- include file: <GL/glu.h>
- GLU routines use the prefix: *glu*

OpenGL Utility Toolkit

- window management
- event management
- window system-independent
- include file: <GL/glut.h>
- GLUT routines use the prefix: *glut*

2.5 OpenGL Command Syntax



OpenGL defined constants begin with GL_, use all capital letters, and use underscores to separate words.

GL_COLOR_BUFFER_BIT
GL_POINTS
GL_LINES
GL_POLYGON
GL_LINE_STRIP
GL_LINE_LOOP

OpenGL Suffix Data Types

Suffix	Data type	Typical C or C++ type	OpenGL type name
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating point	float	GLfloat, GLclampf
d	64-bit floating point	double	GLdouble, GLclampd
ub	8-bit unsigned number	unsigned char	GLubyte, GLboolean
us	16-bit unsigned number	unsigned short	GLushort
ui	32-bit unsigned number	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

* use OpenGL defined data types throughout your application to avoid mismatched types when porting your code between different implementations.

2.6 What do they do?

```
void myInit(void) {  
    ...  
    glMatrixMode (GL_PROJECTION);  
    glLoadIdentity ();  
    gluOrtho2D (0.0, 640.0, 0.0, 480.0);  
    // Establishing a simple coordinate system  
}
```

```
void myDisplay(void) {  
    ...  
    ...  
    glFlush ();  
    // Force execution of the above commands  
}
```

```
int main(int argc, char** argv) {  
    ...  
    glutDisplayFunc(myDisplay);  
    glutMouseFunc(myMouse);  
    glutMainLoop();  
    // Draw the initial picture and enter  
    // the (infinite) event-checking loop  
}
```

2.7 Interaction with the Mouse and Keyboard

Callback function registration commands:

- `glutMouseFunc(myMouse)`
- `glutMotionFunc(myMovedMouse)`
- `glutKeyboardFunc(myKeyboard)`

Callback function prototypes:

```
void myMouse( int button, int state, int x, int  
y);
```

```
void myMovedMouse(int mouseX, int  
mouseY);
```

```
void myKeyboard(unsigned char theKey, int  
mouseX, int mouseY);
```

Generating a Curve by Dragging the Mouse

```
class GLintPoint {
public:
    GLint x, y;    };

void myMouse(int button, int state, int x, int y) {
    switch (button) {
        case GLUT_RIGHT_BUTTON:
            if (state == GLUT_DOWN)    exit (-1);
            break;
        default:
            break;
    }
}

void myMovedMouse(int mouseX, int mouseY) {
    GLint  x = mouseX;
    GLint  y = screenHeight - mouseY;
    GLint  brushSize = 20;
    glColor3f(1.0, 0.0, 0.0);    // set the drawing color to red
    glRecti(x, y, x+brushSize, y+brushSize);
    glFlush();
}
```

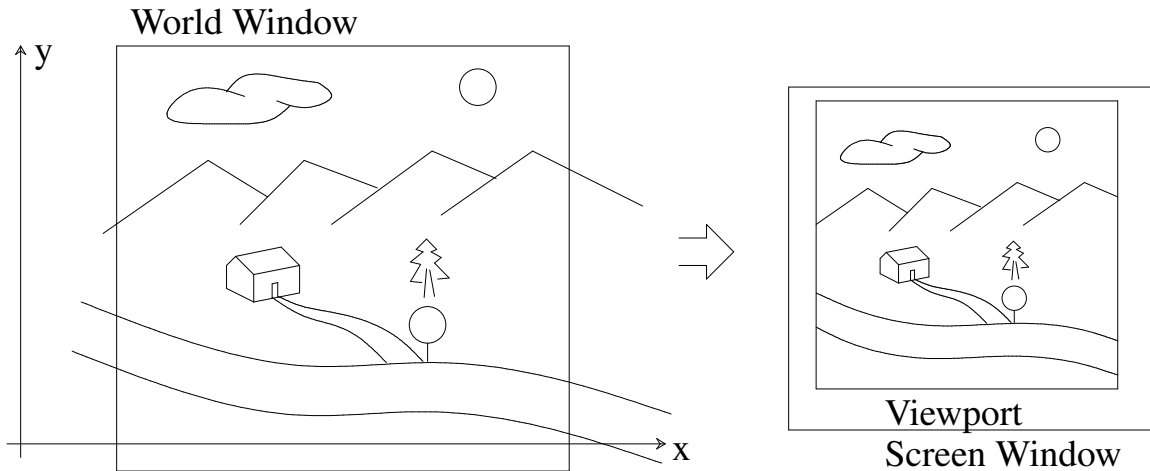
```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    // initialize the toolkit
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    // set display mode
    glutInitWindowSize (screenWidth, screenHeight);
    // set screen window size
    glutInitWindowPosition (100, 150);
    // set window position on screen
    glutCreateWindow (argv[0]);
    // open the screen window
    myInit ();
    glutDisplayFunc(myDisplay);
    // register redraw function
    glutMouseFunc(myMouse);
    // register myMouse function
    glutMotionFunc(myMovedMouse);
    // register myMoveMouse function
    glutMainLoop();
    // go into a perpetual loop
    return 0;
}
```

3. OpenGL - II

3.1 World Coordinate System, World Window, & Viewport

- Using the **device coordinate system** (DCS) directly is not flexible for many applications. Why?
 - Can deal with integers only
 - There is a maximum on the range of x and y
- **Device-independent approach:**
 - Do the drawing in a **World Coordinate System** (WCS)
 - Use **world window** to define the region to be shown
 - Use **viewport** (a rectangular region of the screen window) to show the drawing

Illustration:



- Need a **window-to-viewport** mapping
- The mapping preserves **aspect ratio** ($= \frac{width}{height}$)
- **clipping**: anything that is outside the world window should be discarded before the mapping
- **Clipping** and **mapping** are performed by OpenGL
- Example: plot $sinc(x) = \sin(\pi x)/\pi x$ between $x = -4$ and $x = 4$ in the viewport (0, 640, 0, 480).

Ideal condition: write the code the following way and let the system worry about the mapping (transformation)

```
void myDisplay ( void )
{
    glBegin ( GL_LINE_STRIP );

    for (GLfloat x = -4.0 ; x < 4.0 ; x += 0.1 )
    {
        GLfloat y = sin (3.14159 * x) / (3.14159 * x);

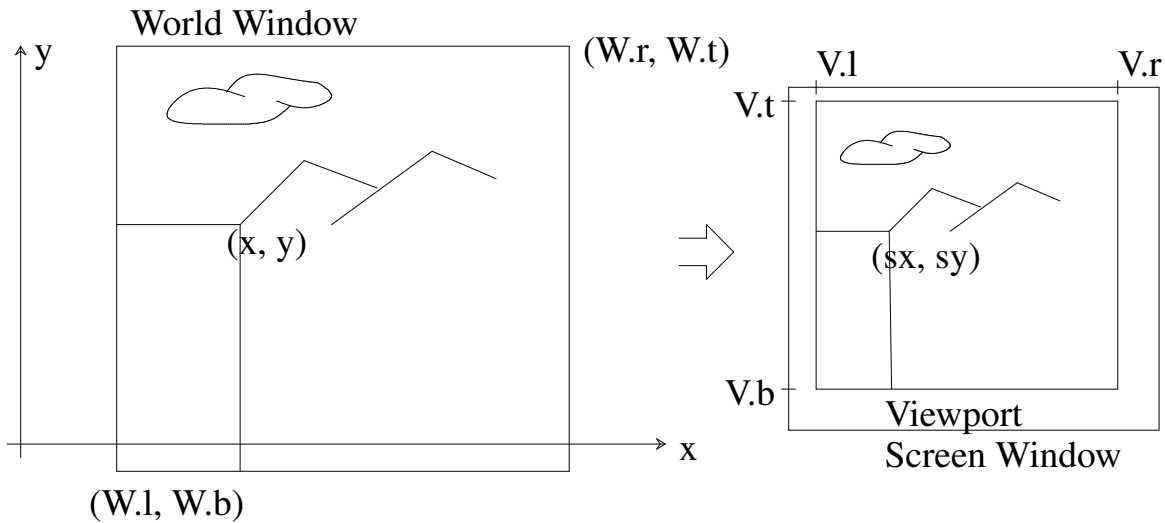
        glVertex2f (x, y);
    }
    glEnd ( );

    glFlush ( );
}
```

How?

Window-to-Viewport Mapping:

- Preserving proportions



$$\frac{sx - V.l}{V.r - V.l} = \frac{x - W.l}{W.r - W.l}$$

and

$$\frac{sy - V.b}{V.t - V.b} = \frac{y - W.b}{W.t - W.b}$$

Hence,

$$\begin{cases} sx = A \cdot x + C \\ sy = B \cdot y + D \end{cases}$$

where

$$A = \frac{V.r - V.l}{W.r - W.l}, \quad C = V.l - A \cdot W.l$$
$$B = \frac{V.t - V.b}{W.t - W.b}, \quad D = V.b - B \cdot W.b$$

Doing it in OpenGL:

Set Window:

```
glMatrixMode ( GL_PROJECTION );  
glLoadIdentity ( ) ;  
gluOrtho2D ( W_left, W_right, W_bottom, W_top );
```

Set Viewport:

```
glViewport ( V_left, V_bottom, V_width, V_height )
```

Example:

```
void myDisplay ( void )
{
glClear ( GL_COLOR_BUFFER_BIT ); // clear the screen
//
glMatrixMode ( GL_PROJECTION );
glLoadIdentity ( );
gluOrtho2D ( -5.0, 5.0, -0.3, 1.0 ); // set the window
//
glViewport(0, 0, 640, 480);    // set the viewport
//
glBegin ( GL_LINE_STRIP );
for ( GLfloat x = -4.0; x < 4.0; x += 0.1 )
{ // draw the plot
    glVertex2f(x, sin(3.14159 * x) / (3.14159 * x));
}
glEnd ( );
glFlush ( );
}
```

3.2 A Few Applications:

1. Tile the screen window

- Use a different viewport for each instance of the pattern

```
/**
void myDisplay(void)
{
glClear ( GL_COLOR_BUFFER_BIT );

glMatrixMode ( GL_PROJECTION );
glLoadIdentity ( );
gluOrtho2D (-5.0, 5.0, -0.3, 1.0 );

for (int i=0; i < 10; i++)
    for (int j=0; j < 11; j++) {
        glViewport ( i*64, j*44, 64, 44);
        // Redraw the plot
        glBegin ( GL_LINE_STRIP );
        for ( GLfloat x = -4.0; x < 4.0; x += 0.1 )
            glVertex2f ( x, sin(3.14159 * x) / (3.14159 * x ) );
        glEnd ( );
    }
glFlush();
}
```

2. Flip an image up side down

- Simply flip the window up side down

```
/**
void myDisplay ( void )
{
glClear ( GL_COLOR_BUFFER_BIT );
//
setWindow ( -5.0, 5.0, -0.3, 1.0 );
//
for ( int i=0; i < 10; i++ )
    for ( int j=0; j < 11; j++ ) {
        if ( ( i+j)%2 == 0 )
            setWindow ( -5.0, 5.0, -0.3, 1.0 );
        else
            setWindow ( -5.0, 5.0, 1.0, -0.3 );
        glVertex2f ( x, sin(3.14159 * x) / (3.14159 * x));
        glEnd ( );
    }
glFlush ( );
}
```

3. Zooming effect

- hold the viewport but reduce (zoom in) or increase (zoom out) the dimension of the window

```
/**
void myDisplay(void)
{
float cx = 0.0, cy = 0.3;    // center of the window
float H, W = 5.0, aspect = 7.143;
int NumFrames = 200;

glClear(GL_COLOR_BUFFER_BIT);    // clear the screen
setViewport(0, 640, 0, 480);    // set the viewport
for(int frame = 0; frame < NumFrames; frame++)
{
    glClear(GL_COLOR_BUFFER_BIT);    // clear the screen
    W *= 0.995;    // reduce the window width
    H = W / aspect;    // maintain the same aspect ratio
    setWindow(cx - W, cx + W, cy - H, cy + H);
    //set the next window
    drawSincFunc ( );
//  glutSwapBuffers ( );
}
}
```

Problems with this approach

- You get flickering, because some portions of the image can be viewed for only very short period of time.

How to achieve smooth animation?

- Use **double buffering**

How?

(1) use "GLUT_DOUBLE" instead of "GLUT_SINGLE"
in

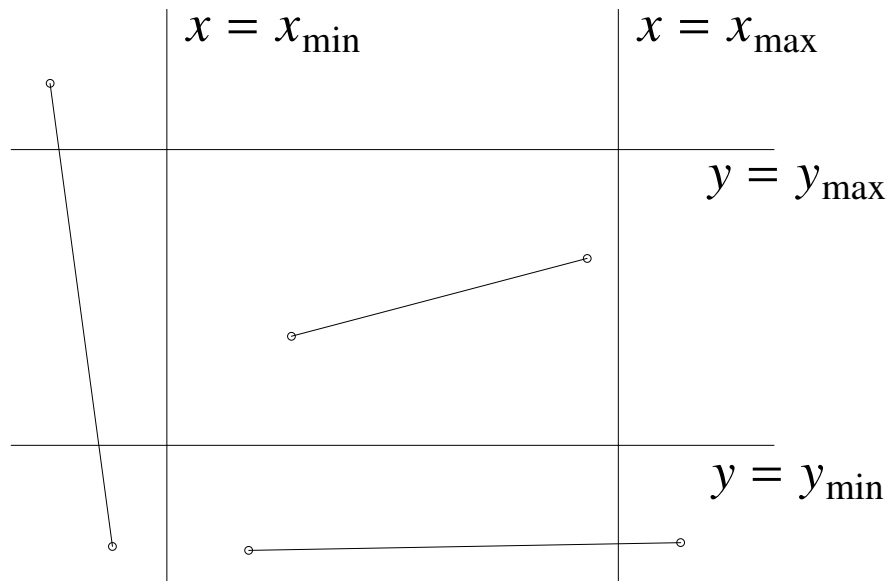
```
glutInitDisplayMode ( xxxx | GLUT_RGB );
```

(2) Include the following instruction at the end of "myDisplay ()".

```
glutSwapBuffers ( );
```

Line clipping: (Cohen-Sutherland algorithm)

- To avoid unnecessary computation, perform tests on trivially accepted cases and trivially rejected cases first
- If both endpoints are inside the window, then the line segment is inside the window
- If both endpoints are to the left ($x < x_{\min}$), to the right ($x > x_{\min}$), below ($y < y_{\min}$), or above ($y > y_{\min}$) the window, then the line segment is outside the window



To perform the tests efficiently, divide the world coordinate system into 9 regions and assign each of them a four-bit code

1001	1000	1010
0001	0000	0010
0101	0100	0110

bit 4 bit 3 bit 2 bit 1

top	bottom	right	left
-----	--------	-------	------

bit 1: sign bit of $(x - x_{\min})$

bit 2: sign bit of $(x_{\max} - x)$

bit 3: sign bit of $(y - y_{\min})$

bit 4: sign bit of $(y_{\max} - y)$

The Cohen-Sutherland Algorithm

1. Compute the codes for the endpoints of the line segment to be clipped
2. Repeat until the line segment is either trivially accepted or rejected

2.1 [Trivial Acceptance Test]

If bitwise OR of the codes is 0000 (line segment is inside the window), draw the line segment and stop.

3. [Trivial Rejection Test]

If bitwise AND of the codes is not 0000 (line segment is outside the window), discard the line segment and stop.

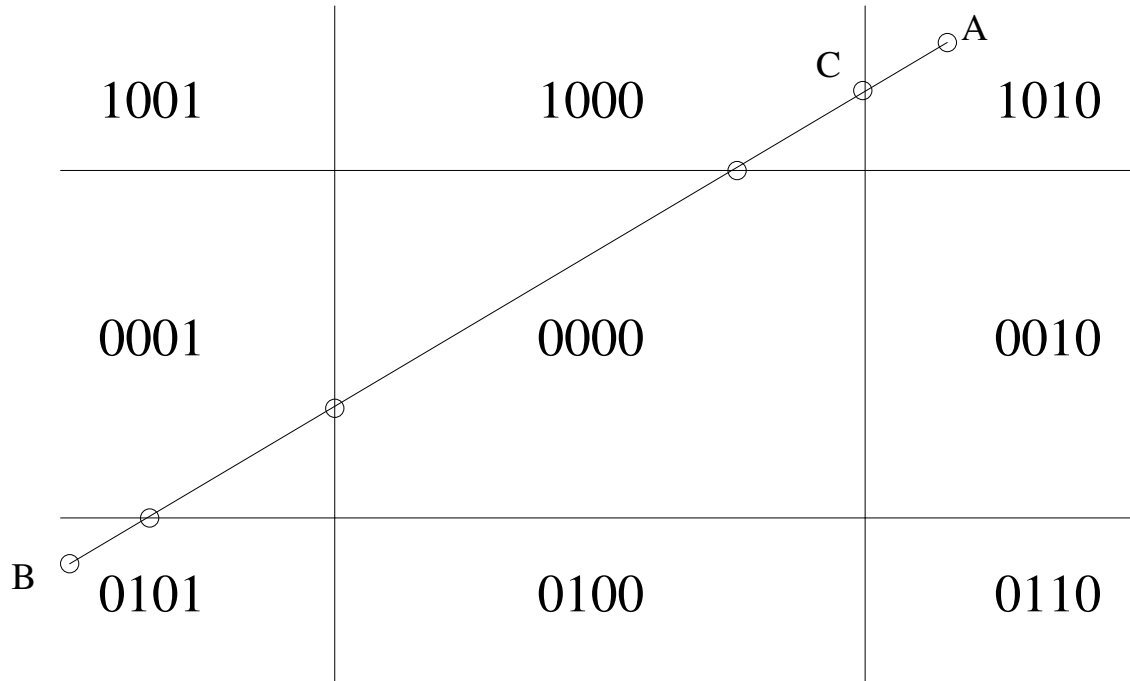
4. [Subdivide the segment]

4.1 Pick an endpoint whose code is non-zero (the endpoint that is outside the window)

4.2 Find the first non-zero bit: this corresponds to the window edge which intersects the line segment

4.3 Compute the intersection point and replace the outside endpoint with the intersection point

An Example



$$\begin{array}{r} \text{A: } 1010 \\ / \text{ B: } 0101 \\ \hline 1111 \end{array}$$

$$\begin{array}{r} \text{A: } 1010 \\ \& \text{ B: } 0101 \\ \hline 0000 \end{array}$$

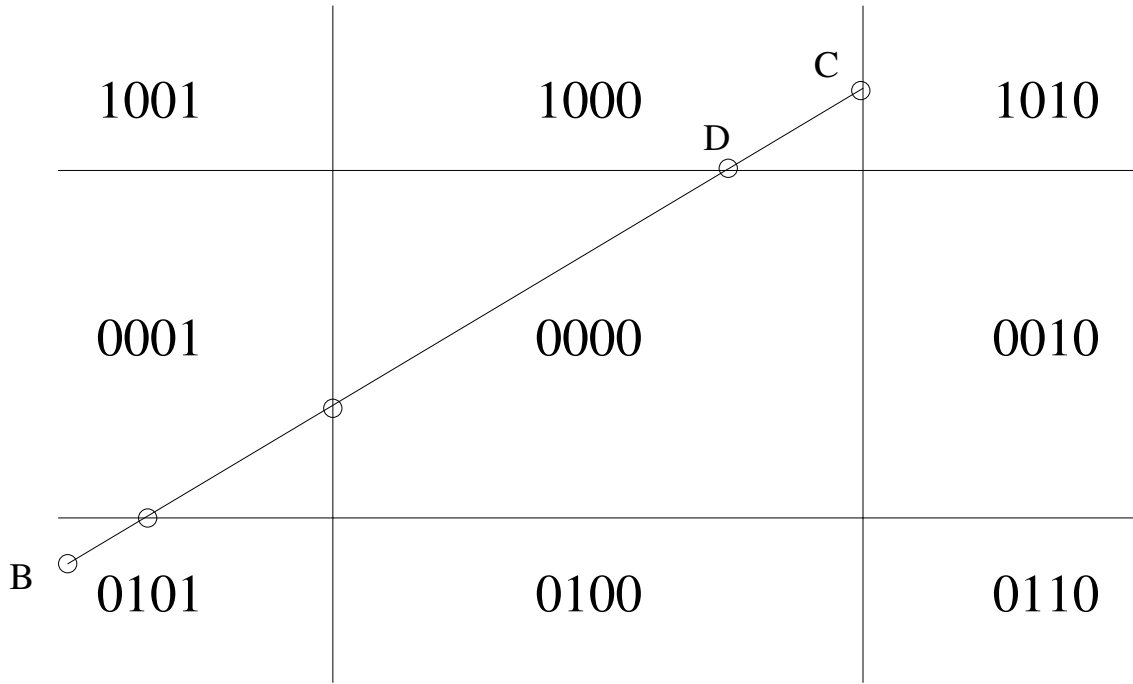
bit 4	bit 3	bit 2	bit 1
top	bottom	right	left

Use bit 2 of *A* (right clipping edge) to do the subdivision

Subdivide at *C* (Find *y* coordinate of *C*)

$$y = m \cdot x_{\max} + b$$

Example (con't)



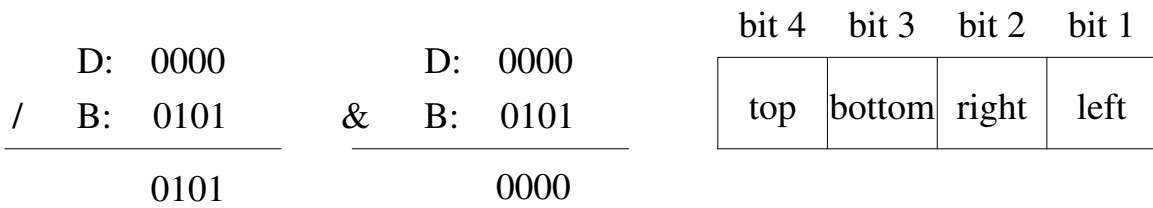
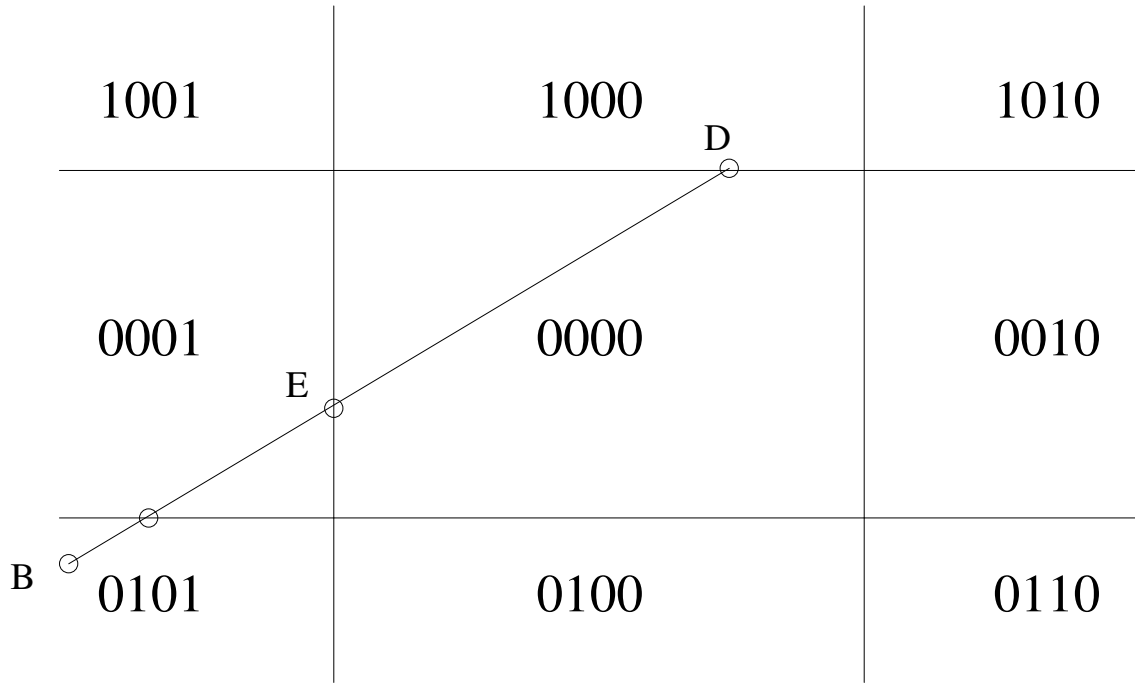
C: 1000	C: 1000	bit 4	bit 3	bit 2	bit 1
/ B: 0101	& B: 0101	top	bottom	right	left
1101	0000				

Use bit 4 of C (top clipping edge) to do the subdivision

Subdivide at D (need to find x coordinate of D)

$$x = (y_{\max} - b)/m$$

Example (con't)

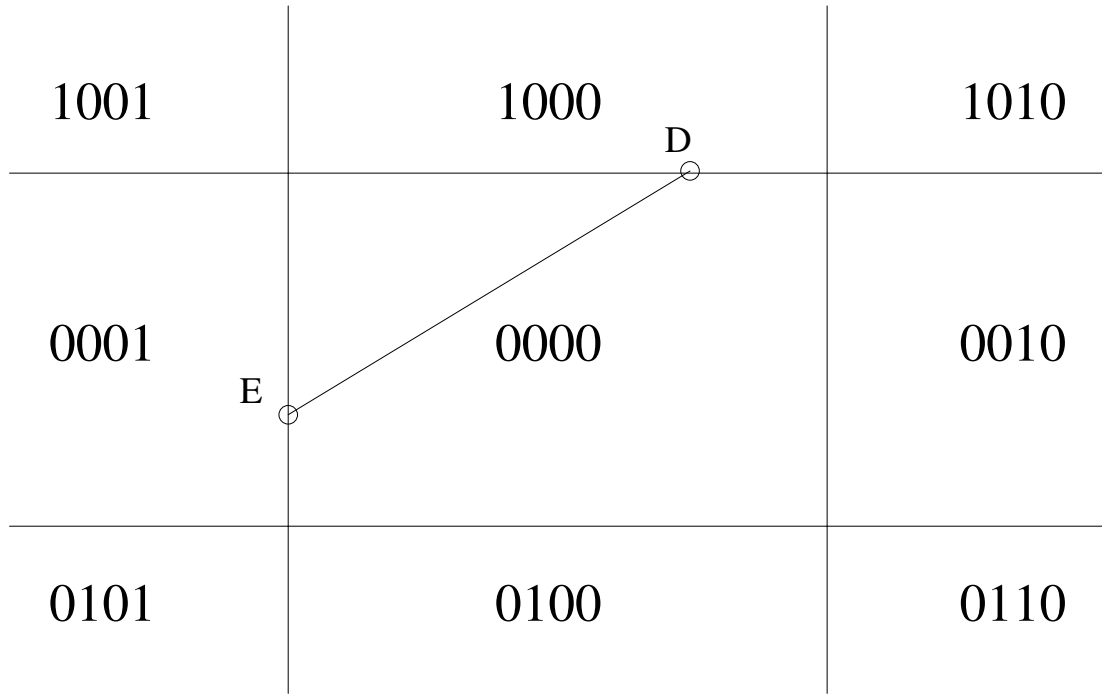


Use bit 1 of B (left clipping edge) to do the subdivision

Subdivide at E (need to find y coordinate of E)

$$y = m \cdot x_{\min} + b$$

Example (con't)



D:	0000	bit 4	bit 3	bit 2	bit 1
/ E:	0000	top	bottom	right	left
<hr/>					
	0000				

Segment ED is trivially accepted