Computer Graphics 2014

3. Introduction to OpenGL

Hongxin Zhang
State Key Lab of CAD&CG, Zhejiang University

2014-09-29

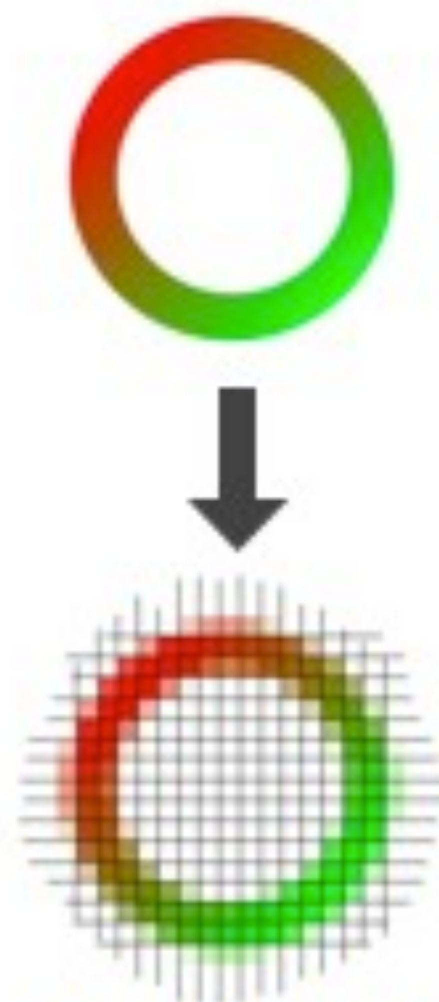
2. 2D Graphics Algorithms (cont.)

Today's Outline

- OpenGL introduction
  - OpenGL primitives
  - Demos / code
- Rasterization rules
- The OpenGL graphics pipeline
- Graphics hardware

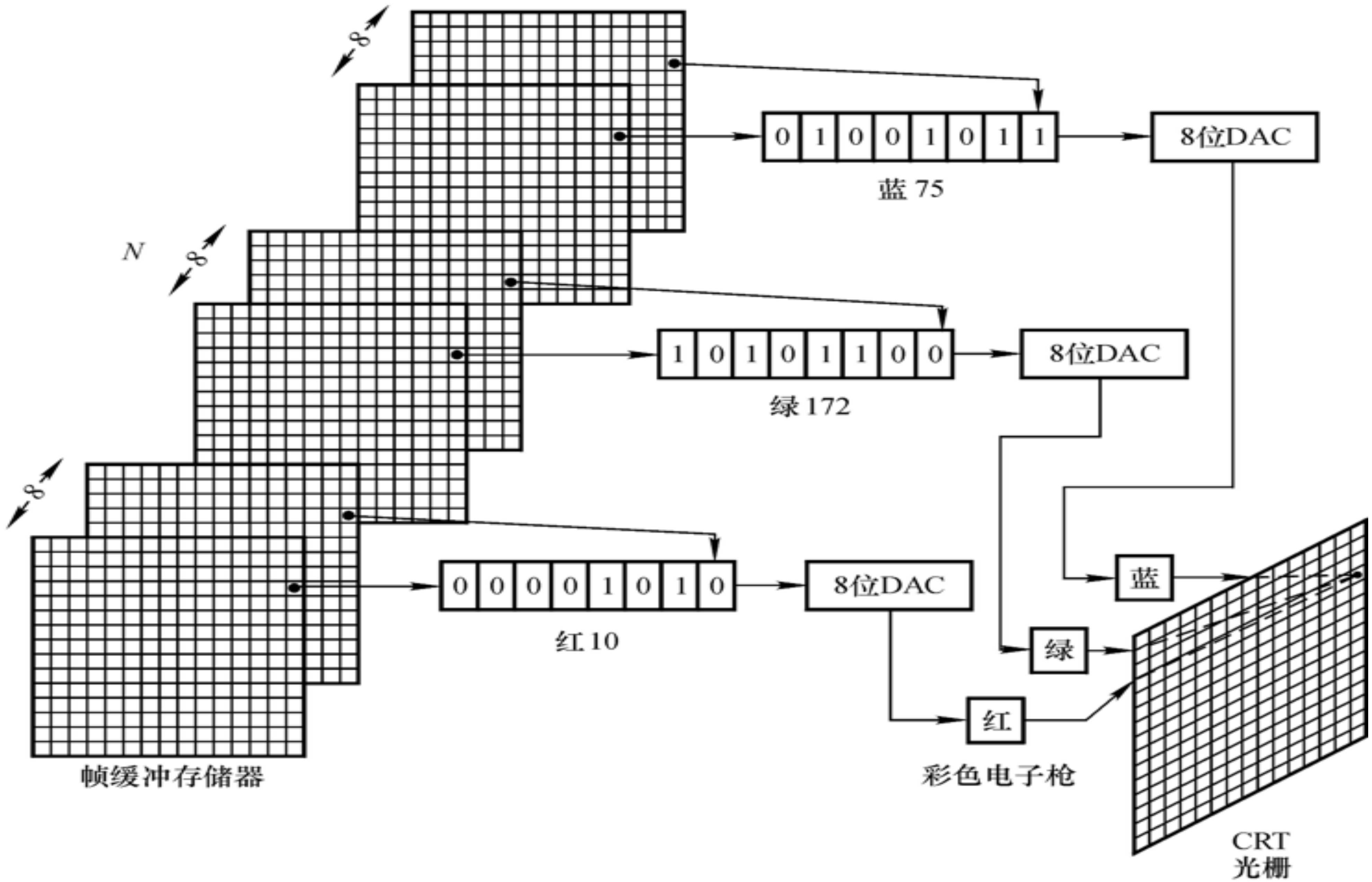
Goal: Understand the graphics pipeline and learn how to create pictures using OpenGL

shapes, lines, points
images, text

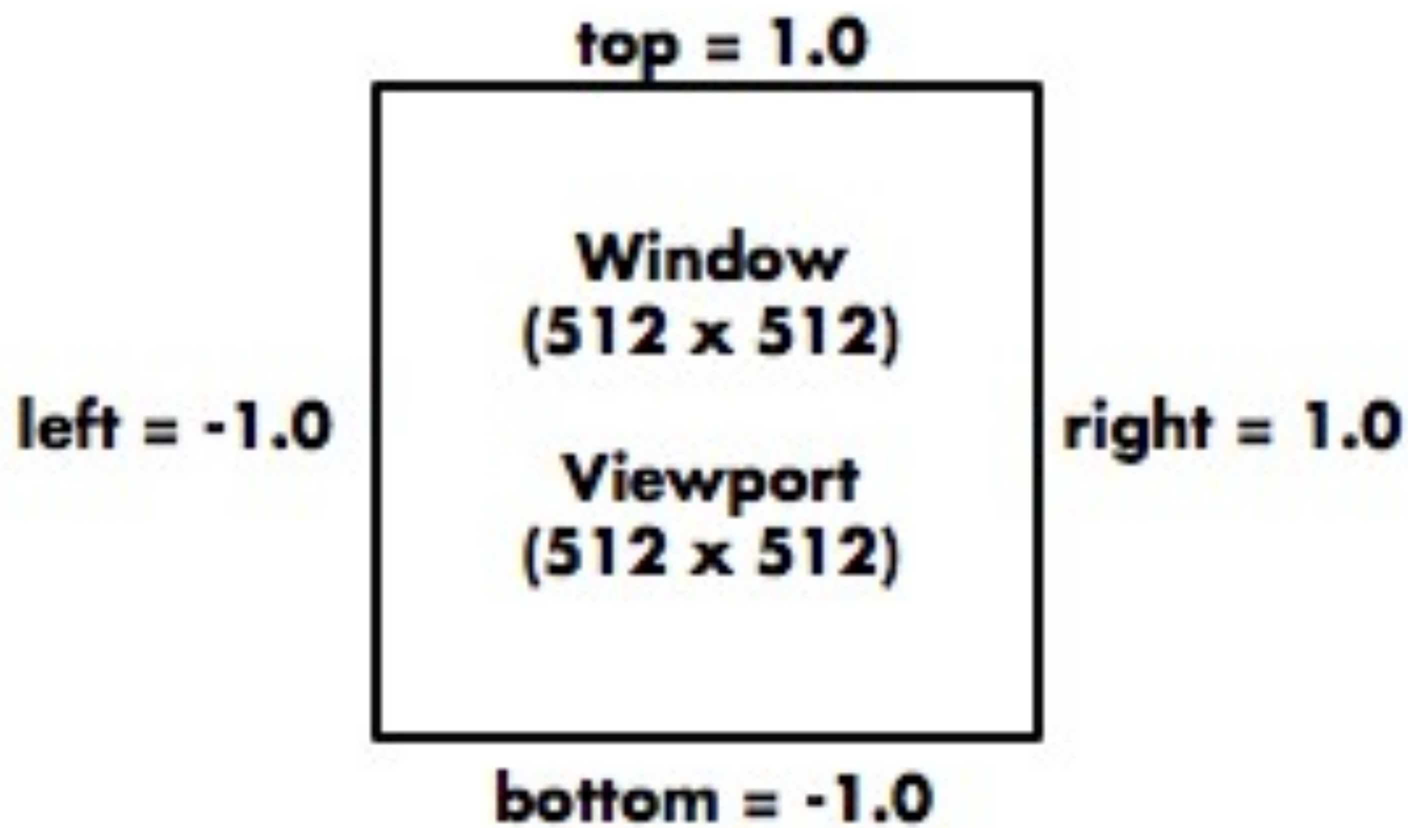


Colored pixels
on screen

Rasterization



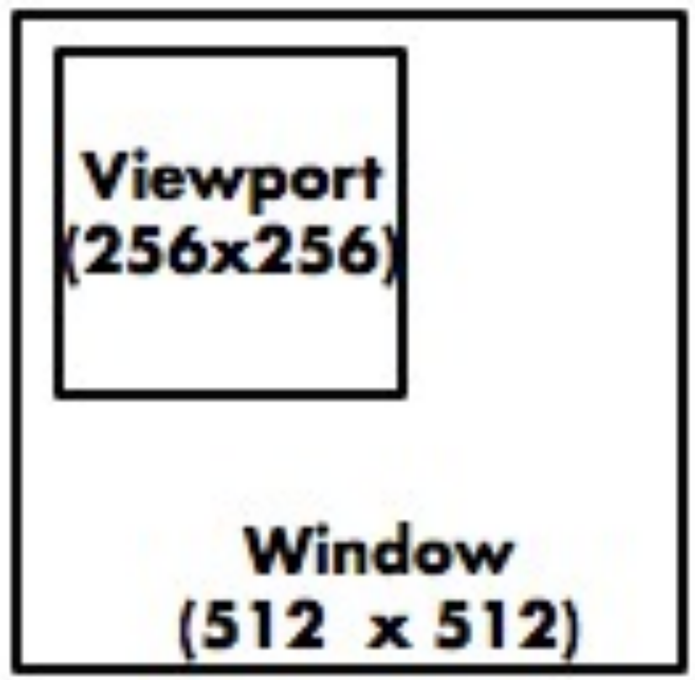
Viewports and Coordinate Systems



Each window has a user coordinate system
A 2D coordinate system is specified by assigning coordinates to the edges of the window
left need not be less than right ...

Framebuffer and Viewport

My Macbook Pro Framebuffer: 1440 x 900



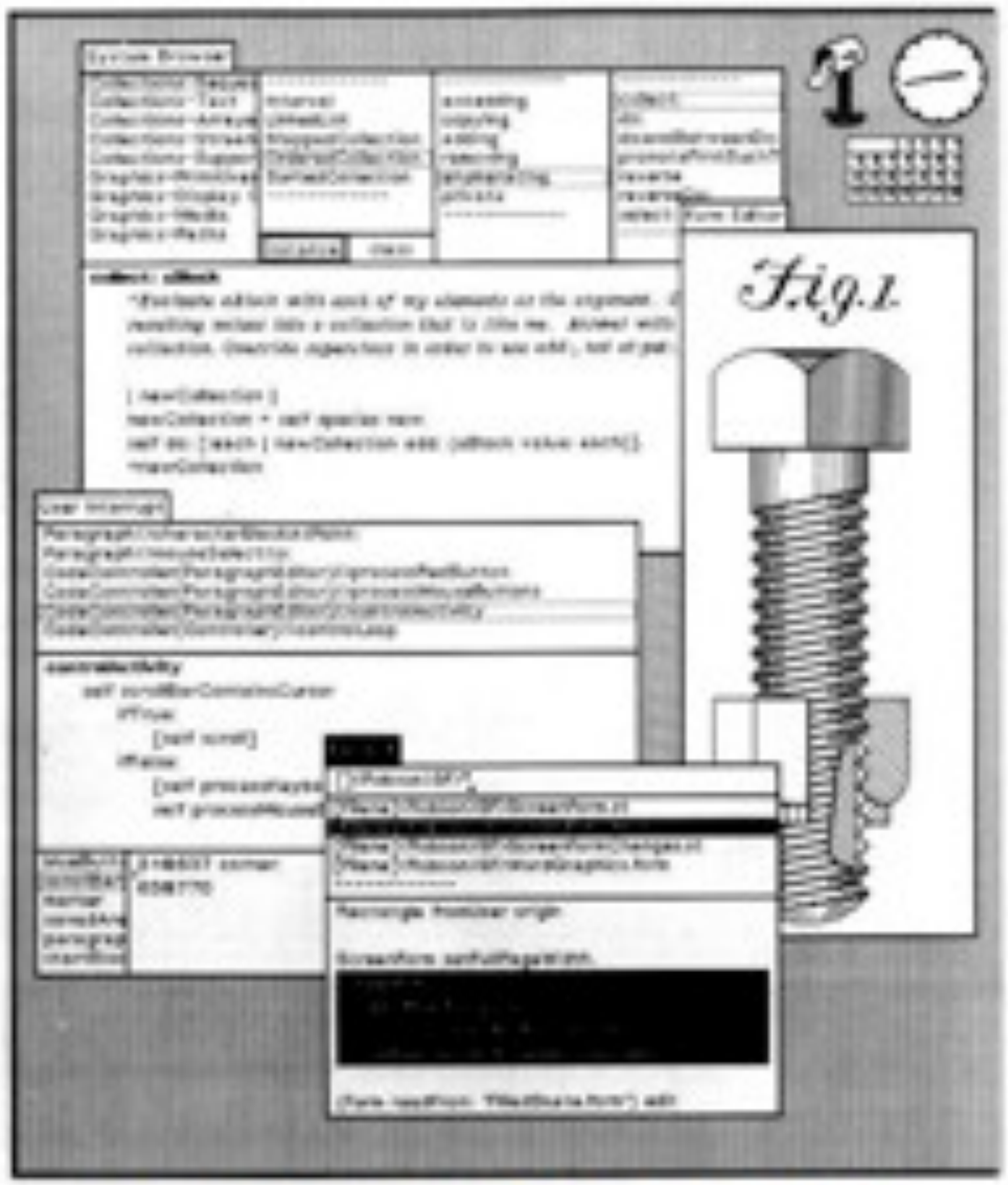
The window is
the portion of the display
usable by the application
(under control of the
"window system")

The viewport is
the portion of the window
that can be drawn in,
no pixels will appear
outside the viewport

All coordinates are integers;
they refer to pixels in the framebuffer

Two Interpretations of Window

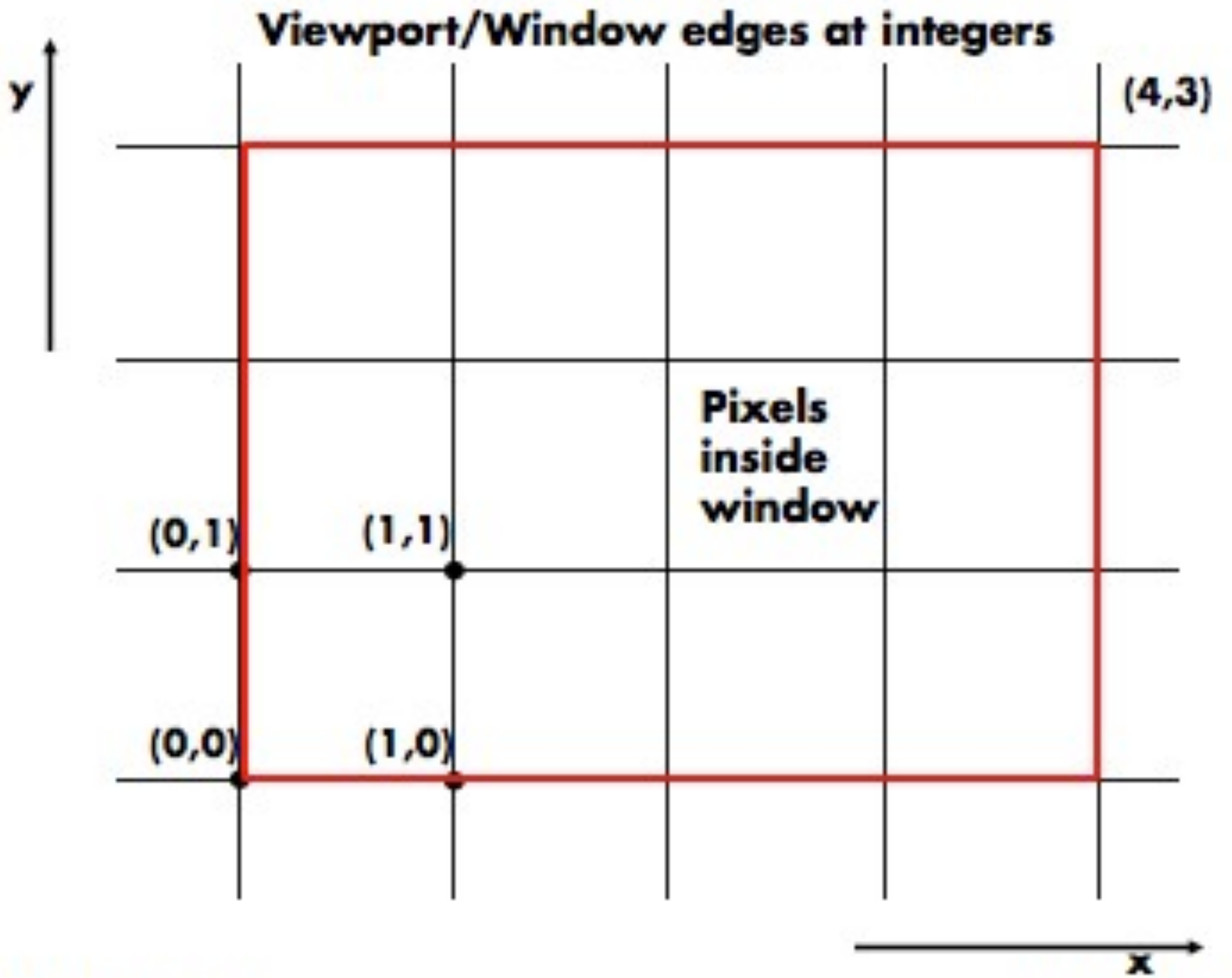
http://www.imaginativeinteriors.co.uk/trompe.shtml



Window on the Display
(Virtual Framebuffer)

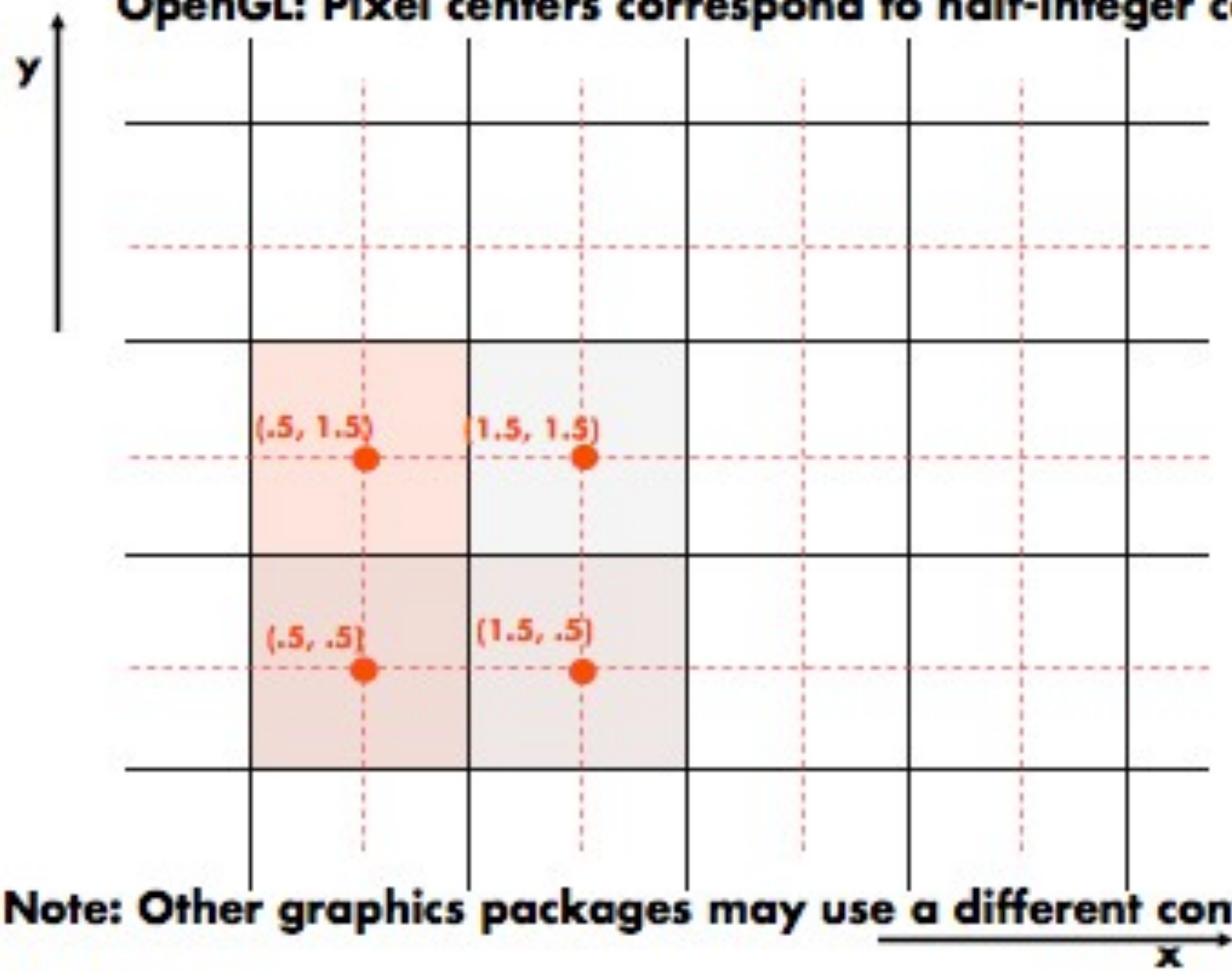
Window into a Virtual World

Pixel Coordinates



Pixel Coordinates

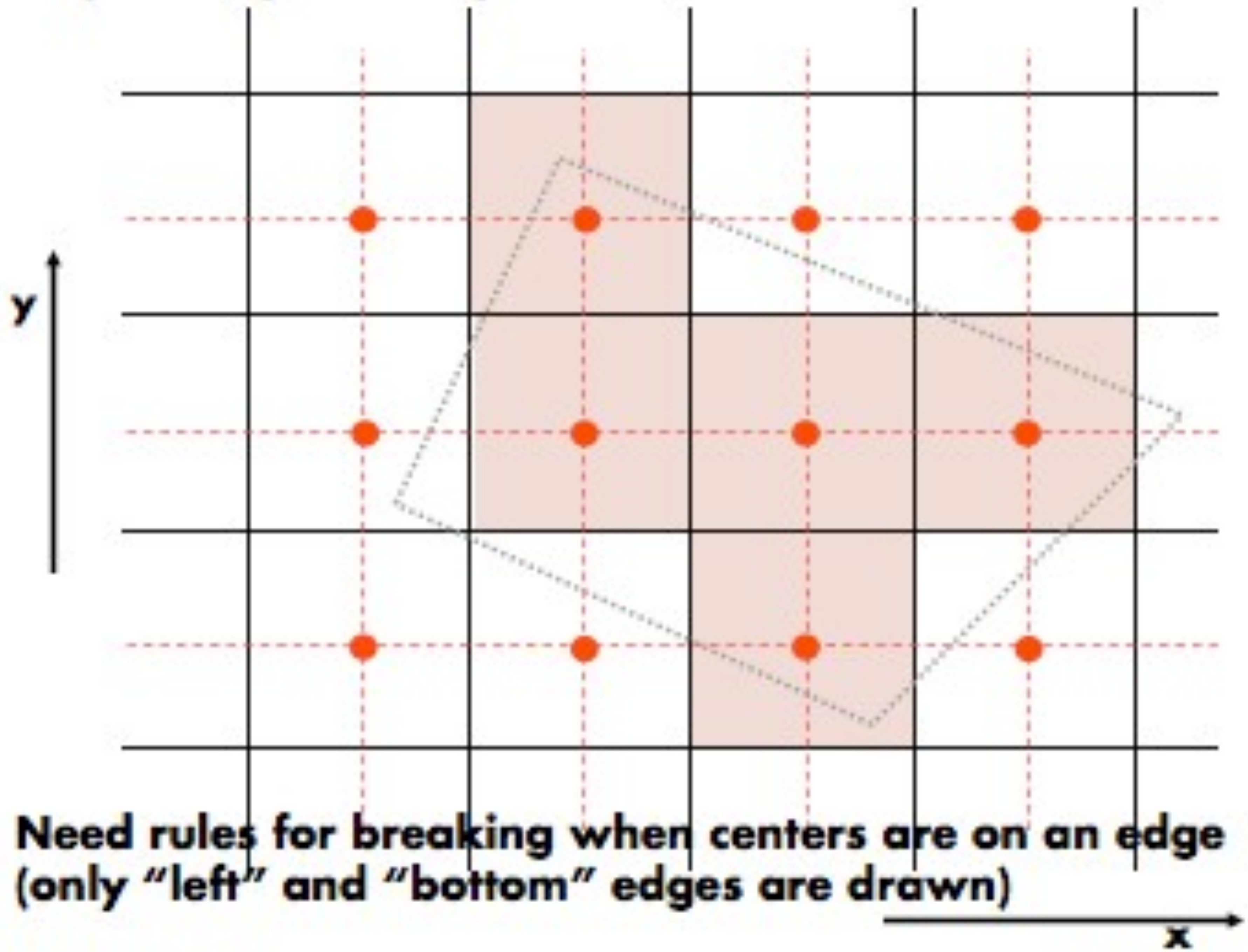
OpenGL: Pixel centers correspond to half-integer coordinates



Note: Other graphics packages may use a different convention

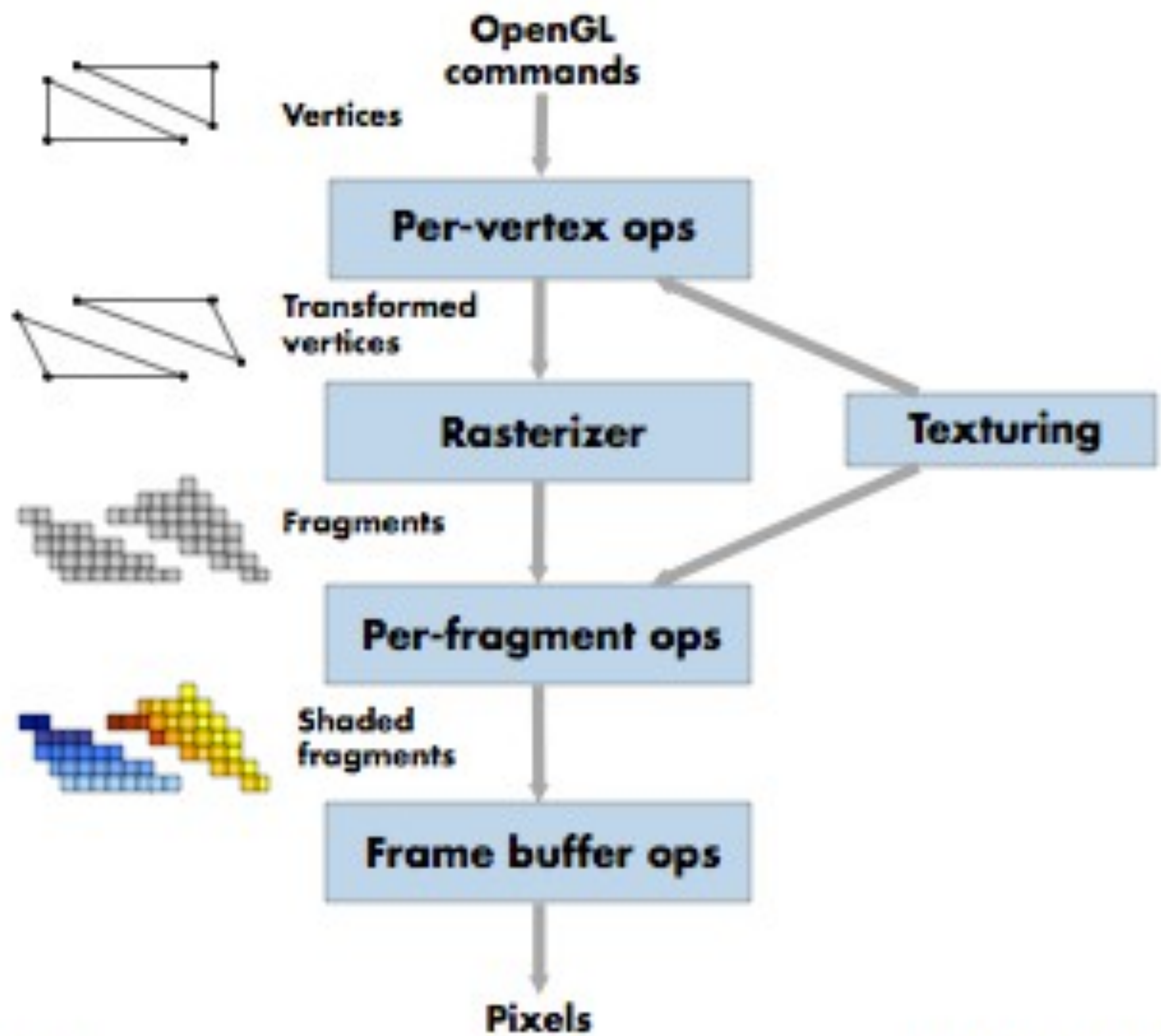
Rasterization Rules: Area Primitives

Output fragment if pixel center is inside area



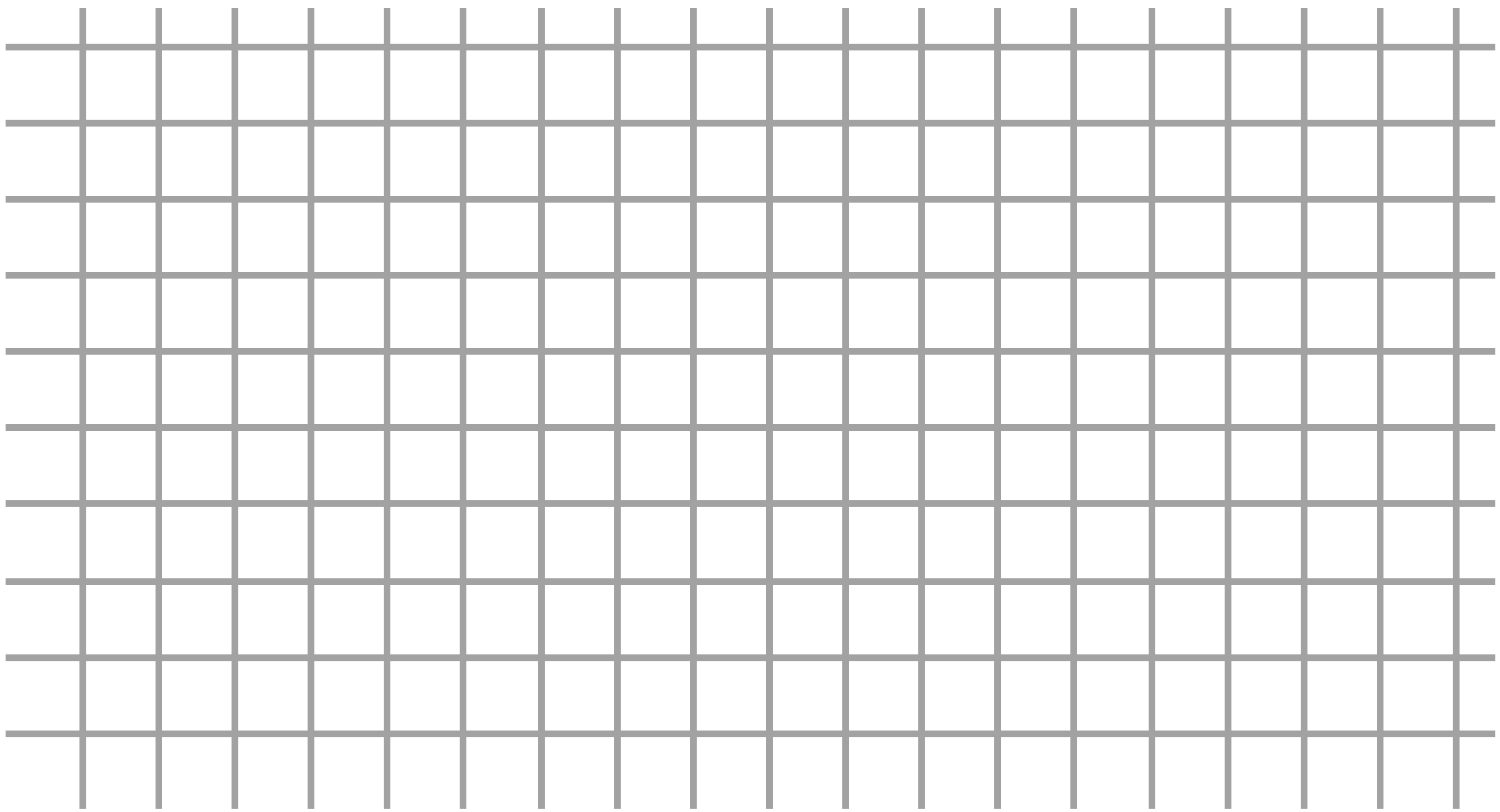
Need rules for breaking when centers are on an edge (only "left" and "bottom" edges are drawn)

Simplified Pipeline



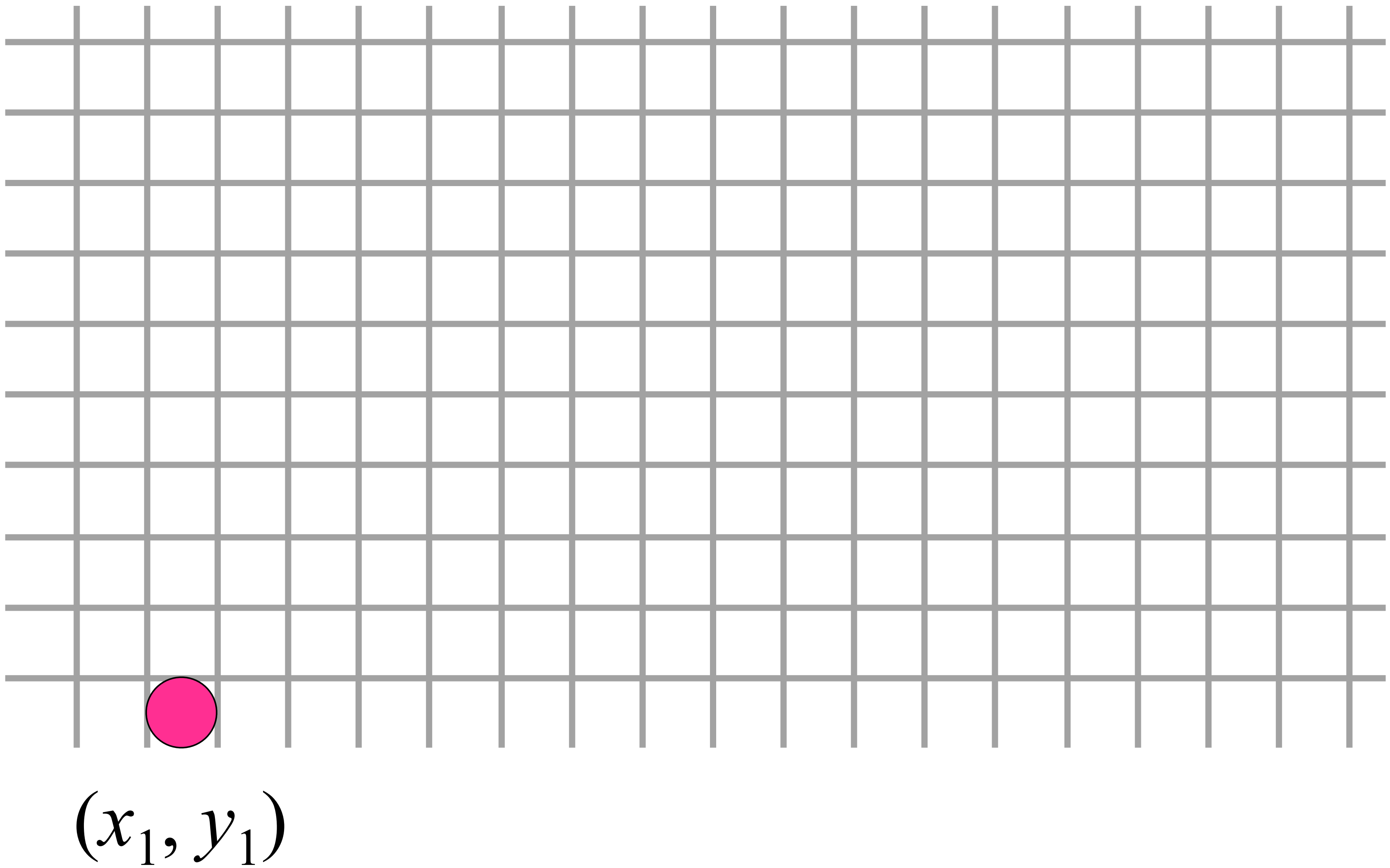
Scan converting lines

start from (x_1, y_1) end at (x_2, y_2)



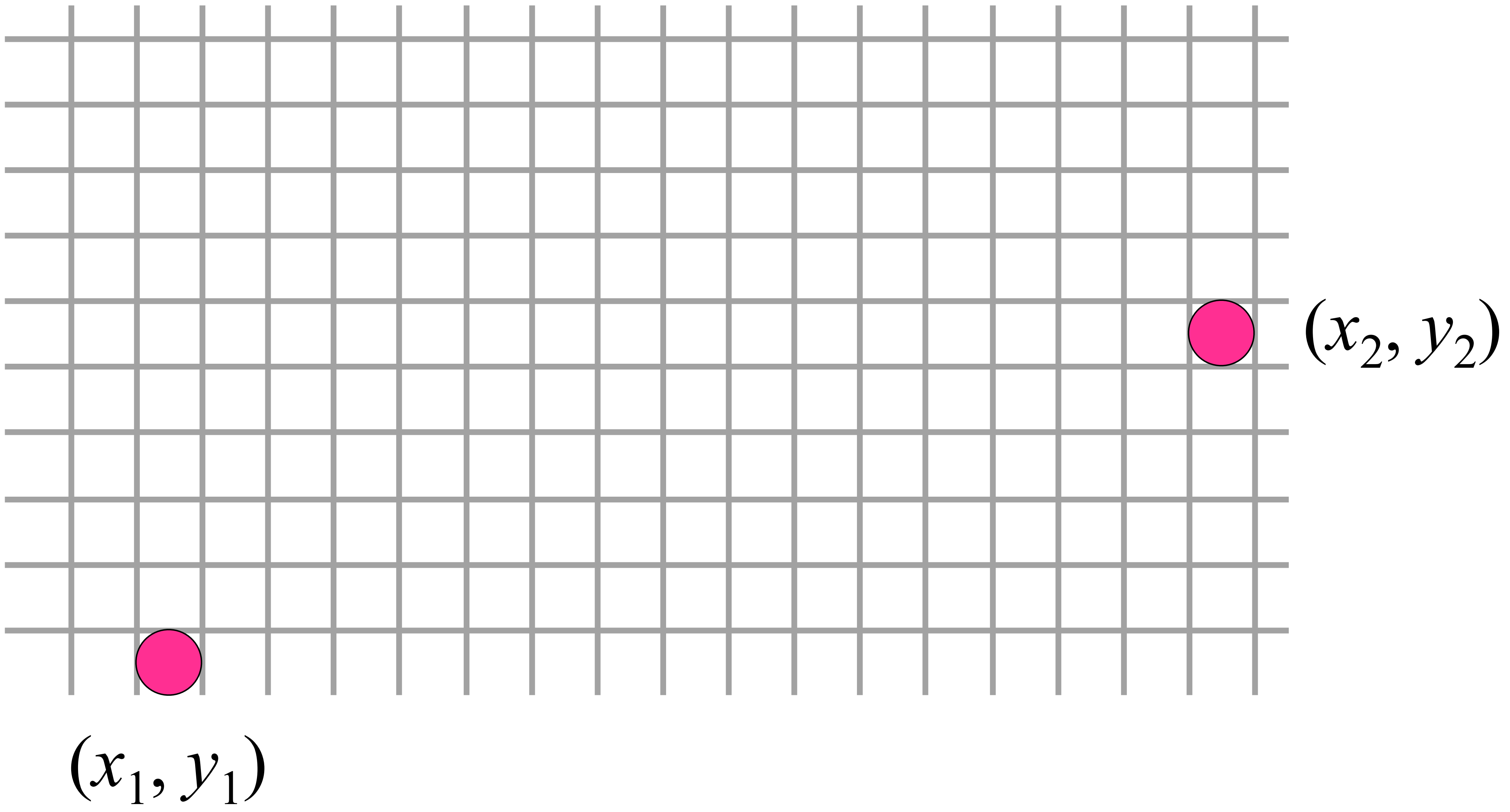
Scan converting lines

start from (x_1, y_1) end at (x_2, y_2)



Scan converting lines

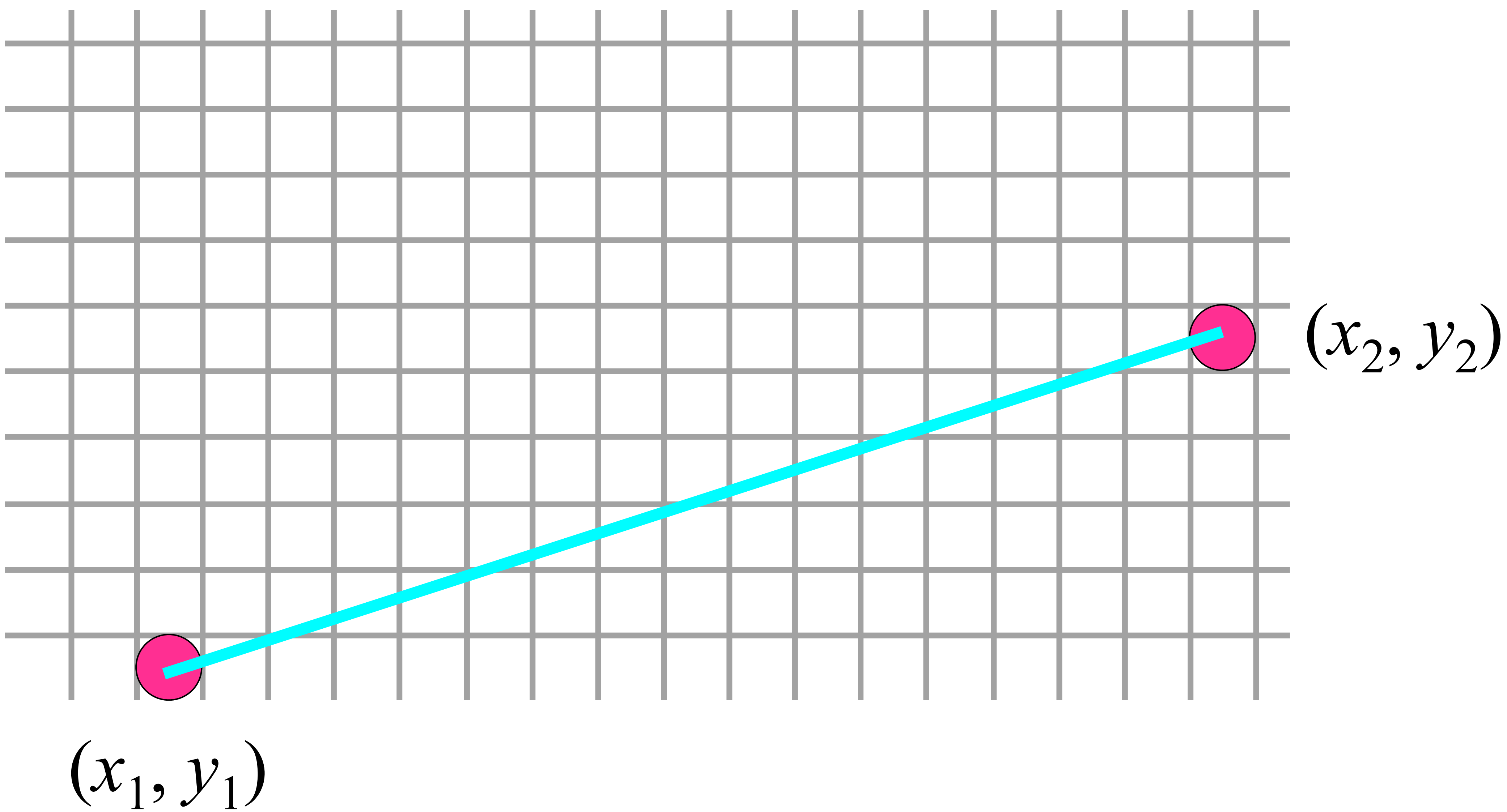
start from (x_1, y_1) end at (x_2, y_2)



Hongxin Zhang, 2014

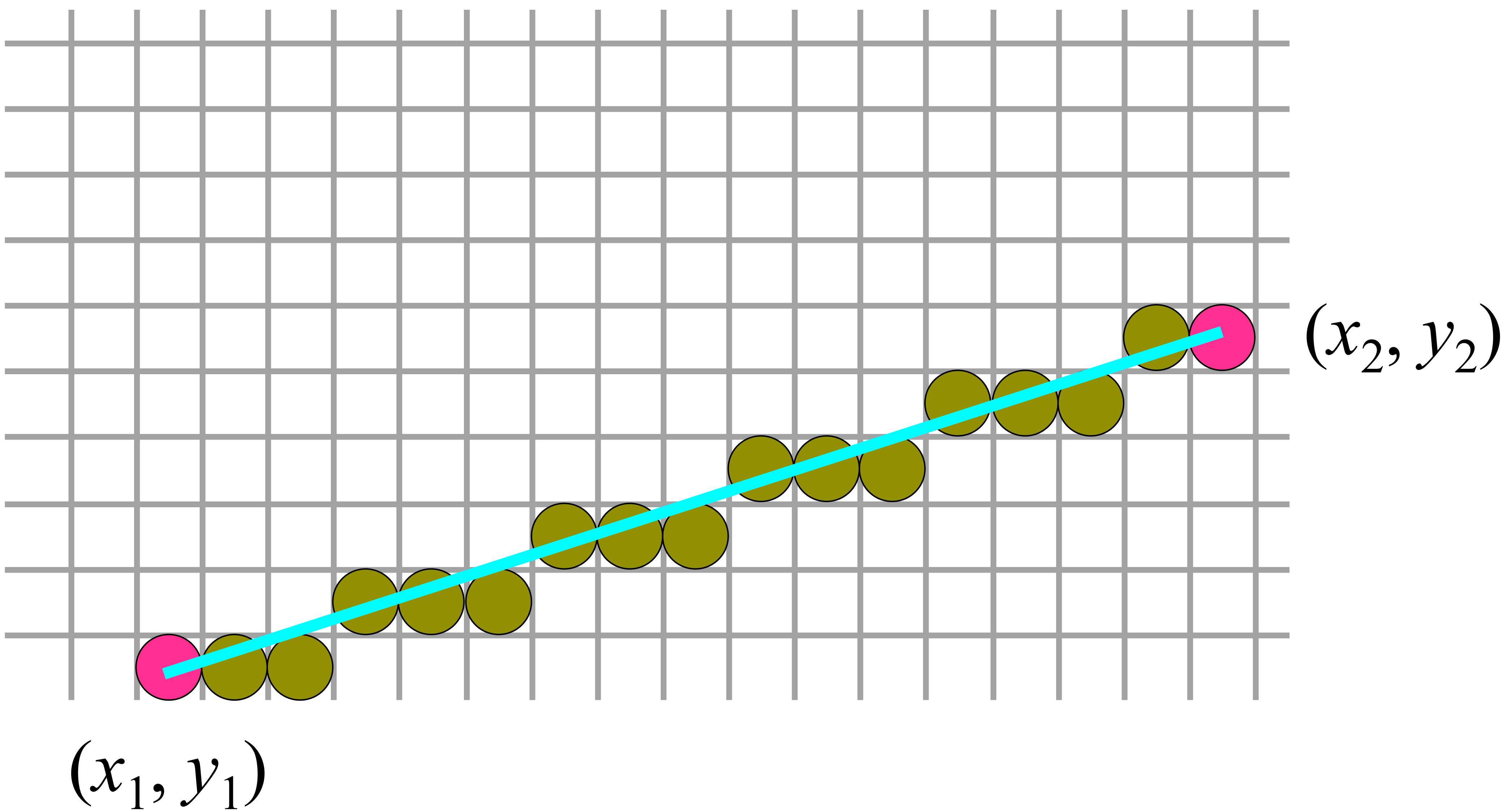
Scan converting lines

start from (x_1, y_1) end at (x_2, y_2)



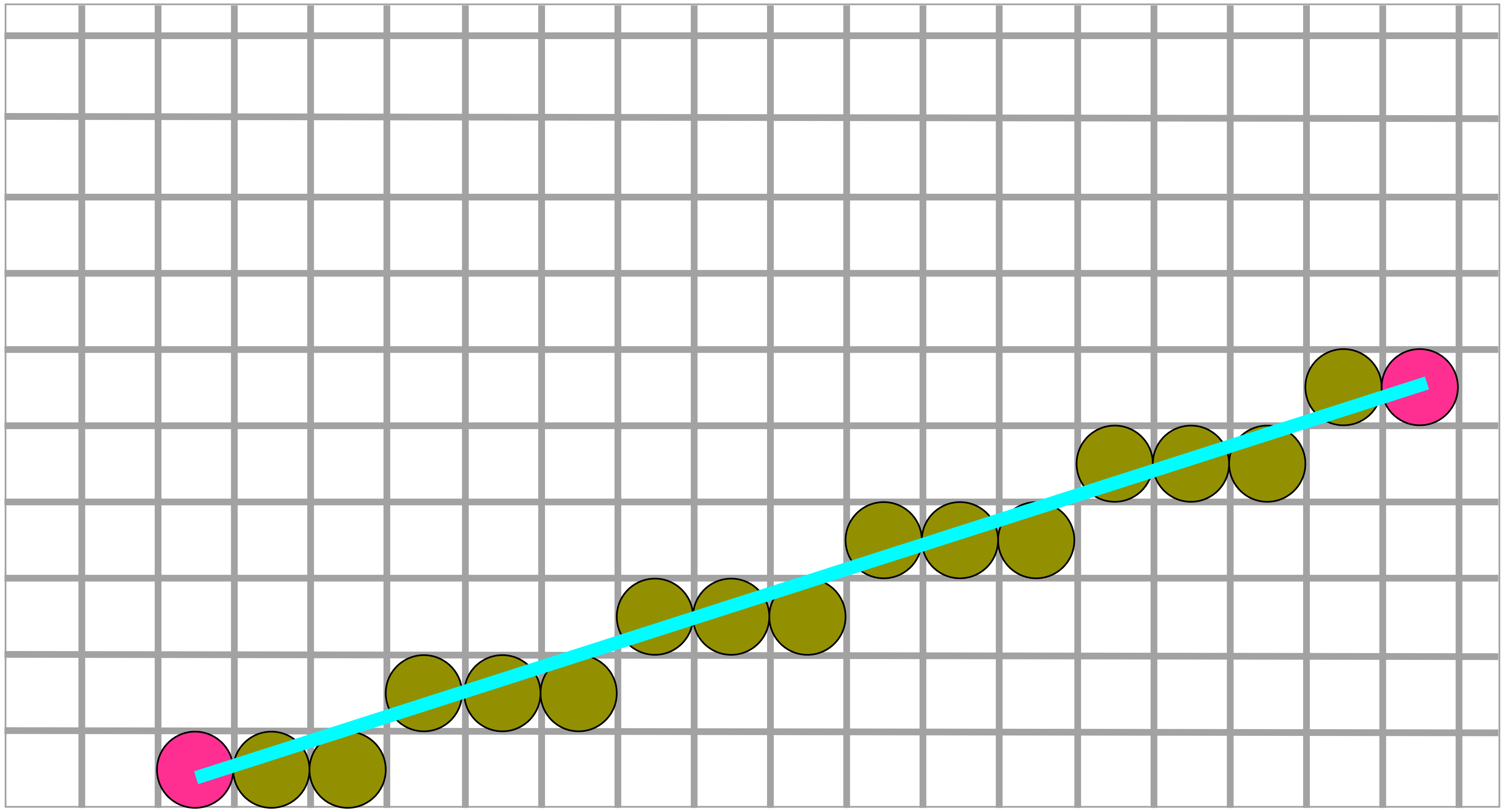
Scan converting lines

start from (x_1, y_1) end at (x_2, y_2)

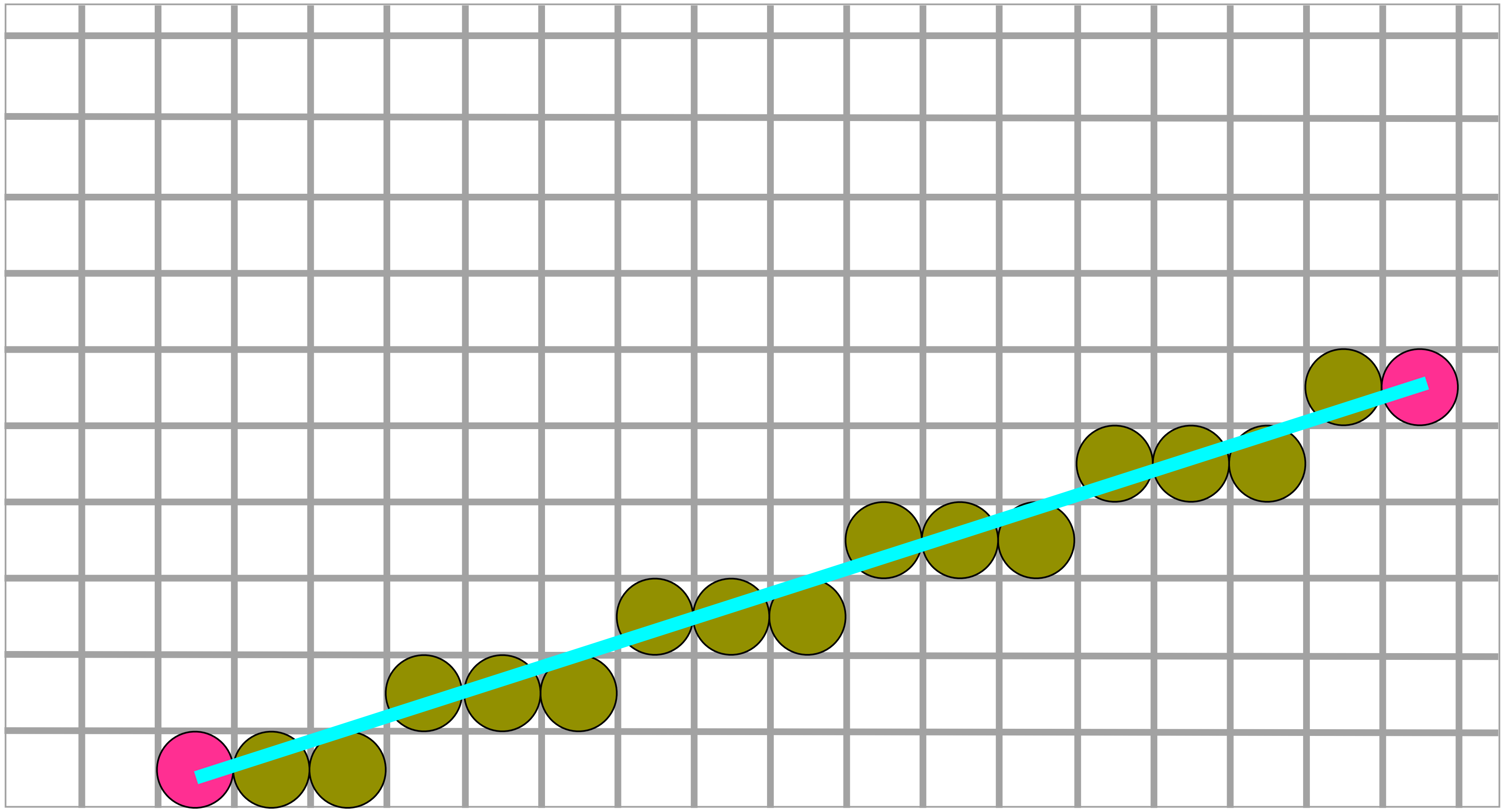


ALG 1. Straightforward

$(x_1, y_1), (x_2, y_2)$



ALG 1. Straightforward



$(x_1, y_1), (x_2, y_2)$

⇓

$y=mx+b$

⇓

$x_1+1 \Rightarrow y=?$, rounding

⇓

$x_1+2 \Rightarrow y=?$, rounding ⟶ $x_1+i \Rightarrow y=?$, rounding

ALG II. Digital Differential Analyzer

- We consider the line in the first octant. Other cases can be easily derived.
- Uses differential equation of the line

$$y_i = mx_i + c$$

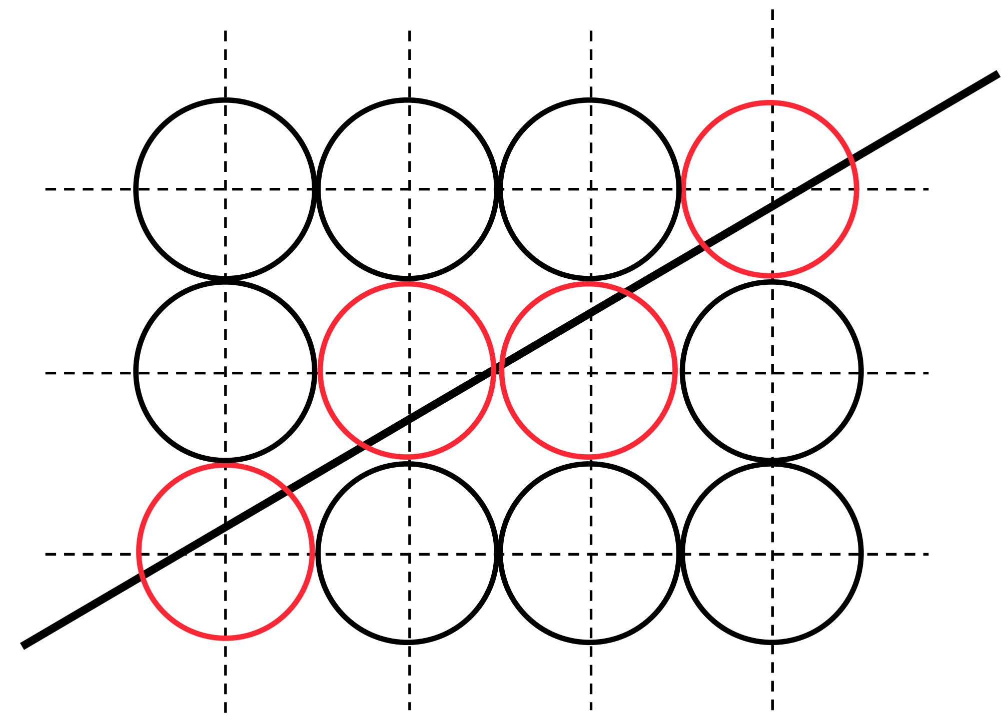
$$where, \quad m = \frac{y2 - y1}{x2 - x1}$$

- Incrementing X-coordinate by 1

$$x_i = x_{i\_prev} + 1$$

$$y_i = y_{i\_prev} + m$$

- Illuminate the pixel $[x_i, round(y_i)]$

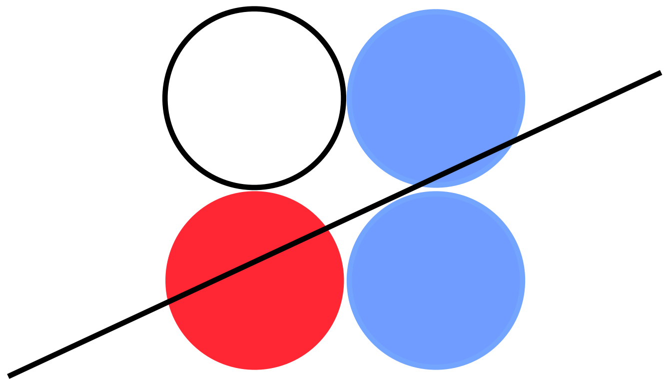


ALG III. Bresenham Line Drawing

$$y_i = mx_i + c$$

$$where, \quad m = \frac{y2 - y1}{x2 - x1}$$

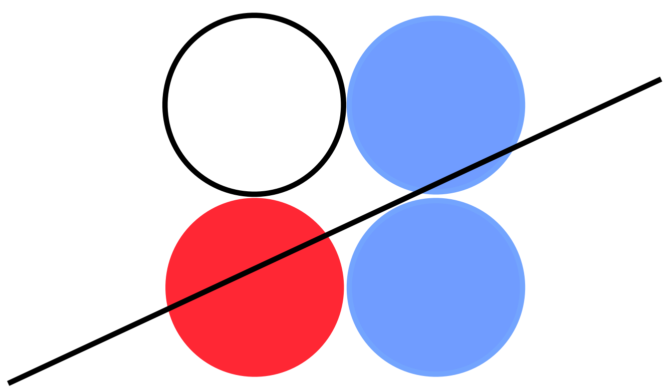
ALG III. Bresenham Line Drawing

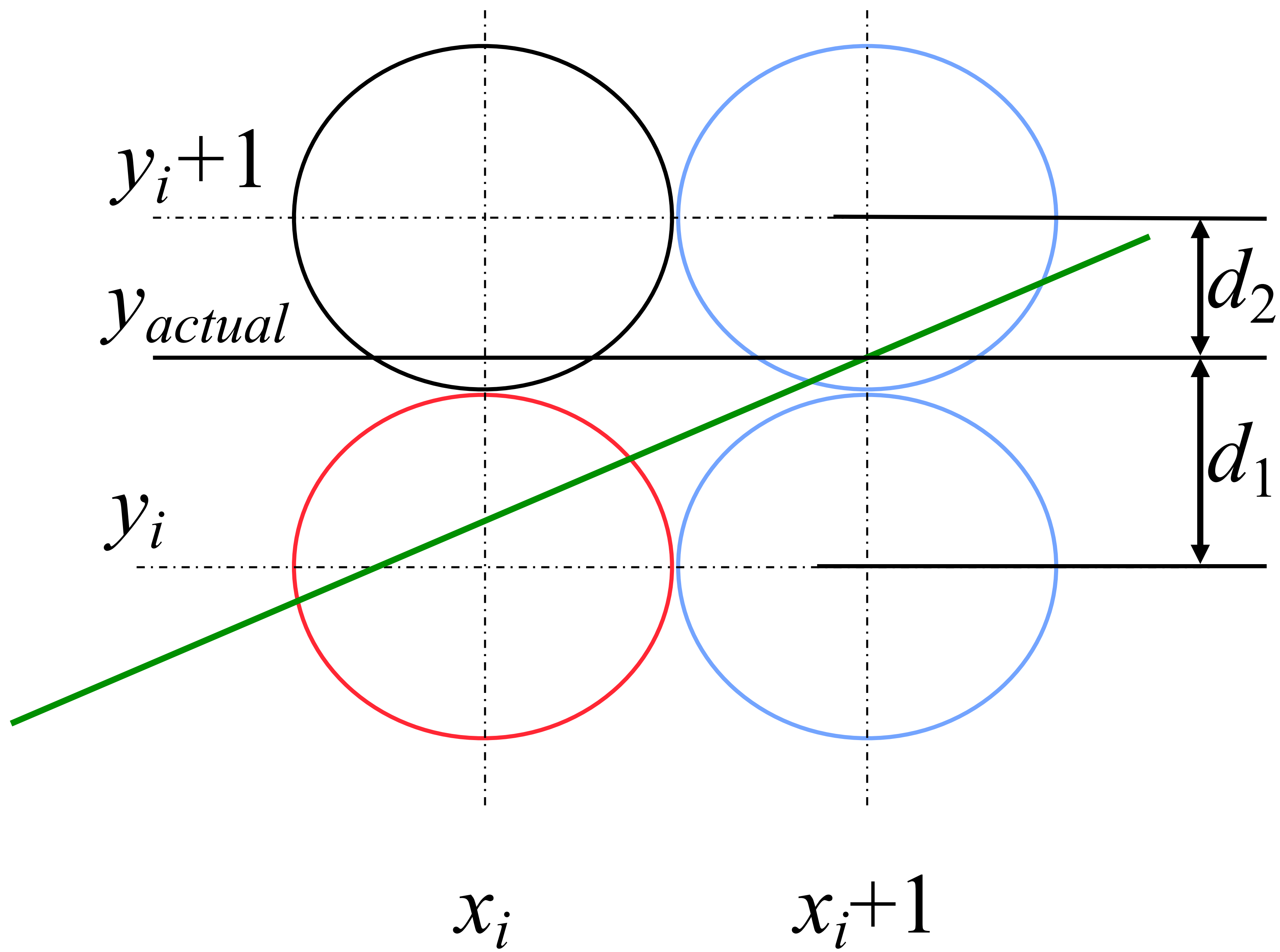


$$y_i = mx_i + c$$

$$where, \quad m = \frac{y2 - y1}{x2 - x1}$$

ALG III. Bresenham Line Drawing

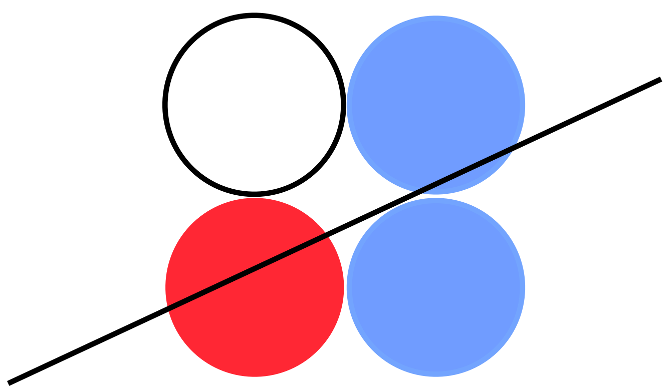


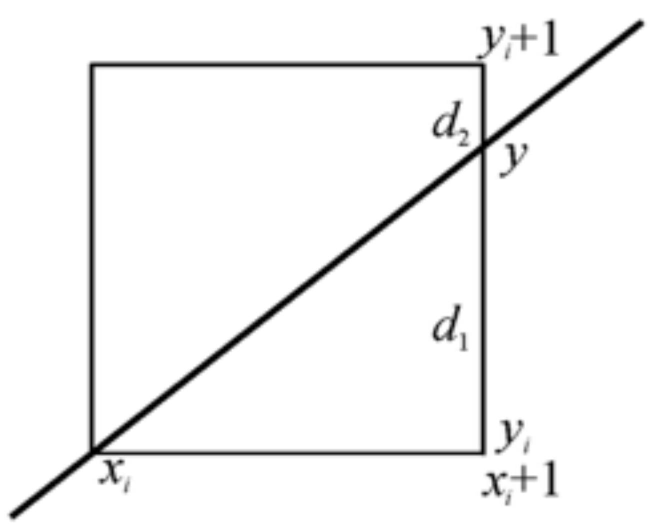


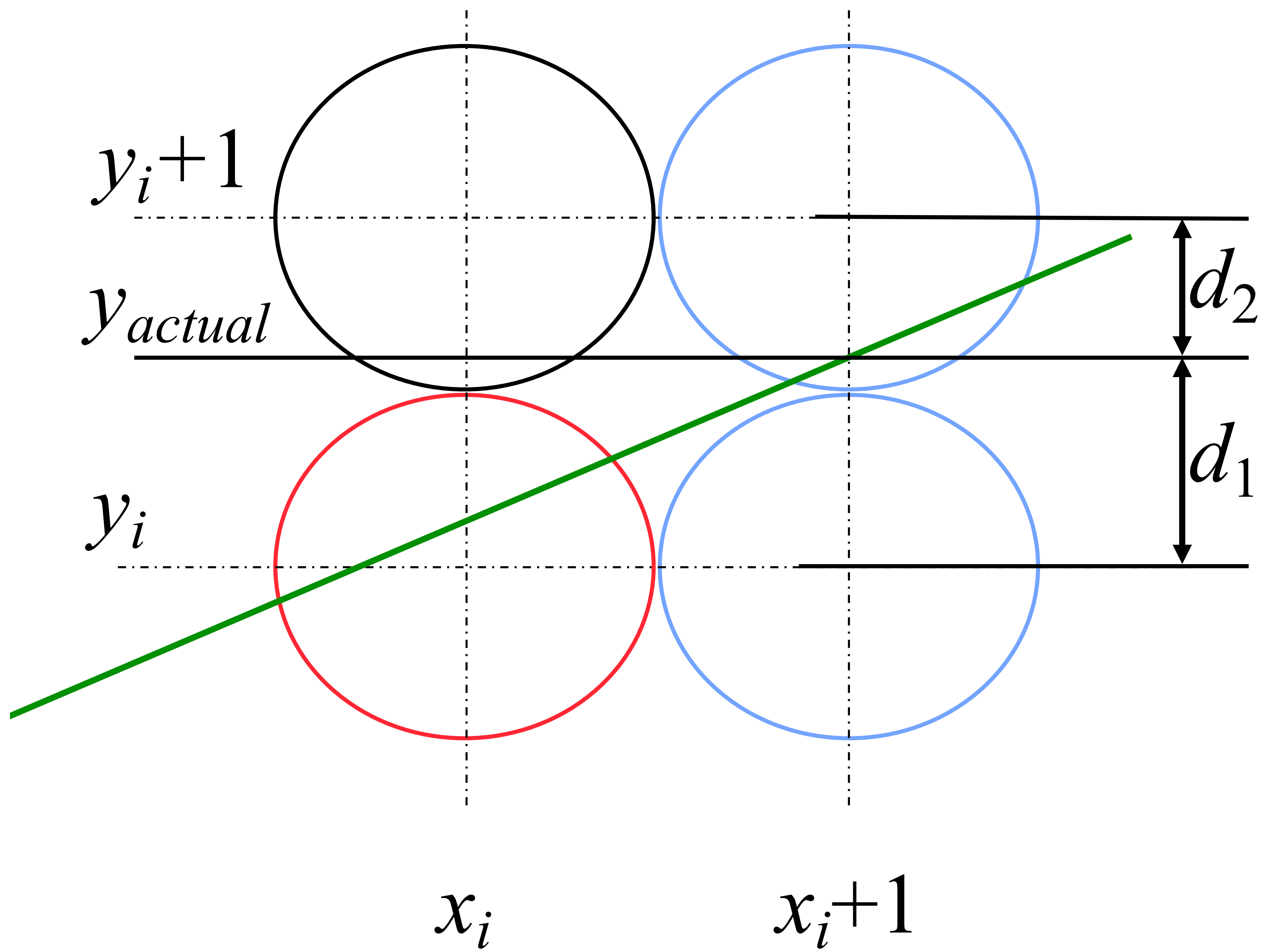
$$y_i = mx_i + c$$

where, $m = \frac{y2 - y1}{x2 - x1}$

ALG III. Bresenham Line Drawing



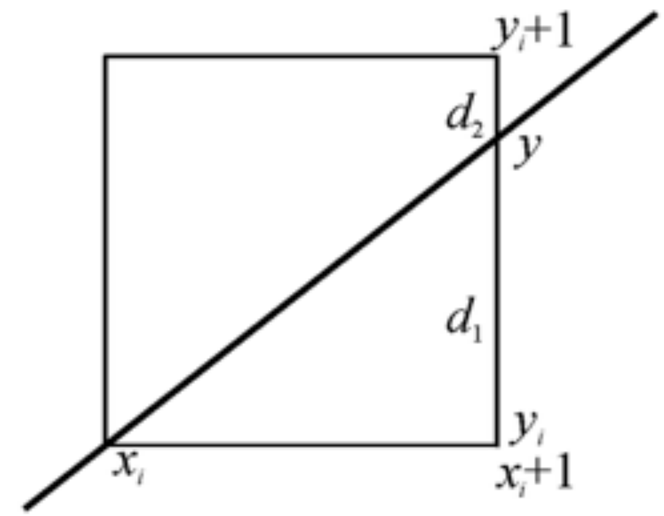




$$y_i = mx_i + c$$

where, $m = \frac{y2 - y1}{x2 - x1}$

$d_1 > d_2$? $\Rightarrow y_{i+1} = y_i$ or $y_{i+1} = y_i + 1$



$y = m(x_i + 1) + b$

(2.1)

$d_1 = y - y_i$

(2.2)

$d_2 = y_i + 1 - y$ (2.3)

If $d_1 - d_2 > 0$, then $y_{i+1} = y_i + 1$, else $y_{i+1} = y_i$

substitute (2.1)、(2.2)、(2.3) into $d_1 - d_2$,

$$d_1 - d_2 = 2y - 2y_i - 1 = 2dy/dx * x_i + 2dy/dx + 2b - 2y_i - 1$$

on each side of the equation, * dx, denote $(d_1 - d_2)$ dx as P_i, we have

$$P_i = 2x_i\,dy - 2y_i\,dx + 2dy + (2b-1)dx \qquad (2.4)$$

Because in first octant $dx > 0$, we have $sign(d_1 - d_2) = sign(P_i)$

If $P_i > 0$, then $y_{i+1} = y_i + 1$, else $y_{i+1} = y_i$

$P_{i+1} = 2x_{i+1}dy - 2y_{i+1}dx + 2dy + (2b-1)dx$, note that $x_{i+1} = x_i + 1$

$$P_{i+1} = P_i + 2dy - 2(y_{i+1} - y_i)\,dx \qquad (2.5)$$

Bresenham algorithm in first octant

1. Initialization $P_0 = 2\,\mathrm{d}y - \mathrm{d}x$
2. draw (x_1, y_1)， $\mathrm{d}x=x_2-x_1$， $\mathrm{d}y=y_2-y_1$，
   Calculate $P_1=2\mathrm{d}y-\mathrm{d}x$， $i=1$；
3. $x_{i+1} = x_i + 1$
   if $P_i>0$， then $y_{i+1}=y_i+1$， else $y_{i+1}=y_i$；
4. draw (x_{i+1}, y_{i+1});
5. calculate P_{i+1}:
   if $P_i>0$ then $P_{i+1}=P_i+2\mathrm{d}y-2\mathrm{d}x$，
   else $P_{i+1}=P_i+2\mathrm{d}y$；
6. $i=i+1$； if $i<\mathrm{d}x+1$ then goto 3； else end

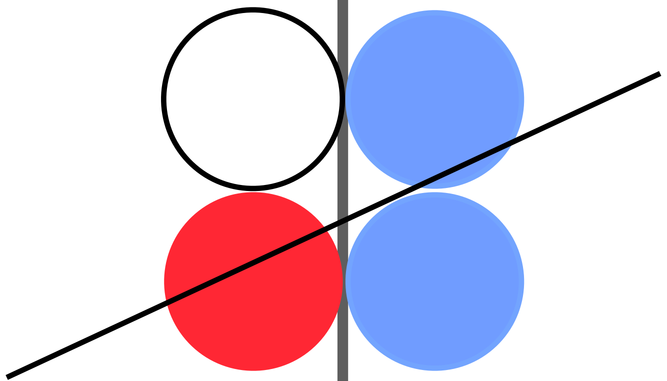
Bresenham algorithm in first octant

1. Initialization $P_0 = 2\,\mathrm{d}y - \mathrm{d}x$
2. draw (x_1, y_1)， $\mathrm{d}x = x_2 - x_1$， $\mathrm{d}y = y_2 - y_1$，
   Calculate $P_1 = 2\mathrm{d}y - \mathrm{d}x$， $i = 1$；
3. $x_{i+1} = x_i + 1$
   if $P_i > 0$， then $y_{i+1} = y_i + 1$， else $y_{i+1} = y_i$；
4. draw (x_{i+1}, y_{i+1});
5. calculate P_{i+1}:
   - if $P_i > 0$ then $P_{i+1} = P_i + 2\mathrm{d}y - 2\mathrm{d}x$，
   - else $P_{i+1} = P_i + 2\mathrm{d}y$；
6. $i = i + 1$； if $i < \mathrm{d}x + 1$ then goto 3； else end

Question: Is it faster than DDA ?

Bresenham algorithm in first octant

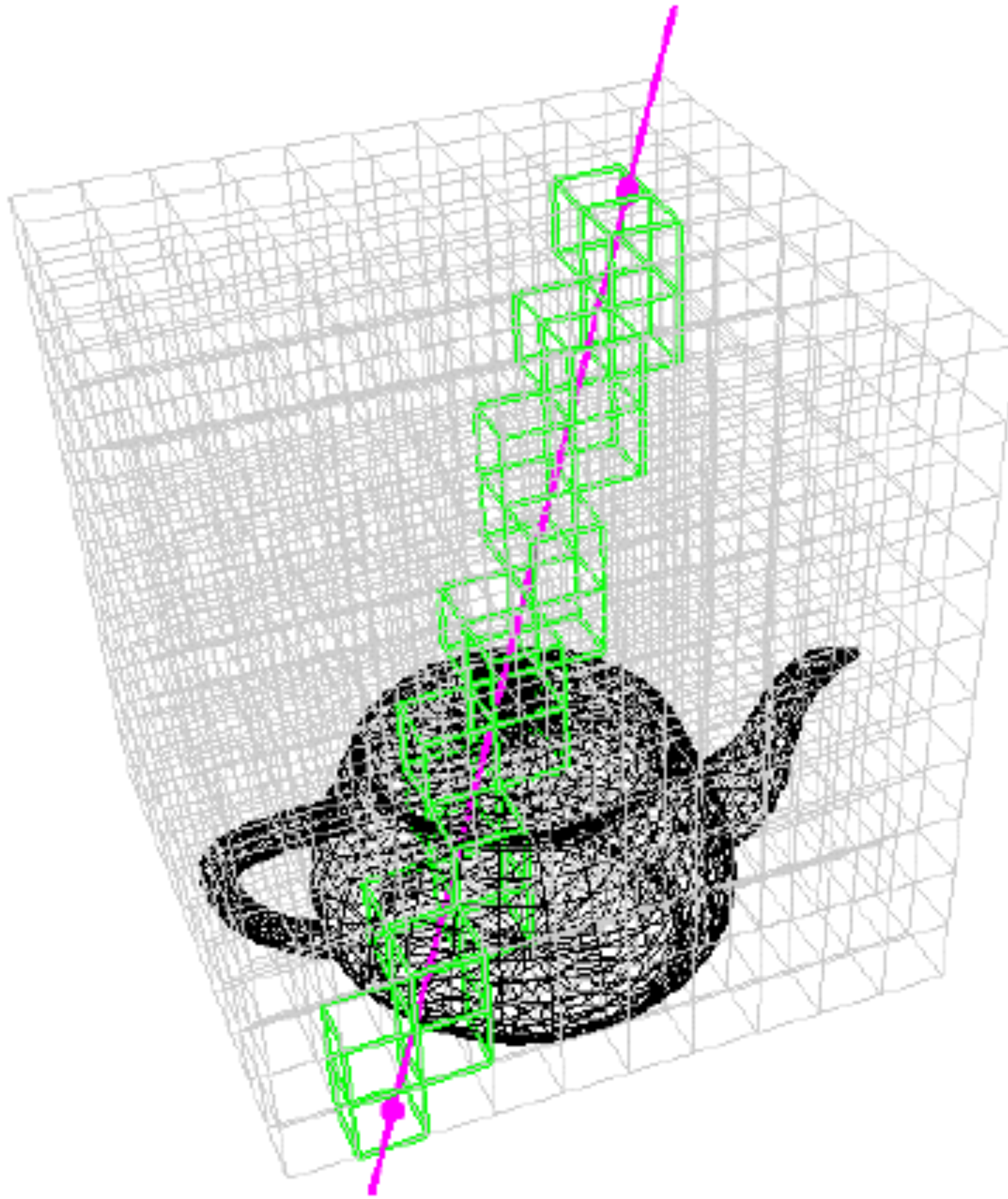
1. Initialization $P_0 = 2\,\mathrm{d}y - \mathrm{d}x$
2. draw (x_1, y_1), $\mathrm{d}x=x_2-x_1$, $\mathrm{d}y=y_2-y_1$,
   Calculate $P_1=2\mathrm{d}y-\mathrm{d}x$, $i=1$;
3. $x_{i+1} = x_i + 1$
   if $P_i>0$, then $y_{i+1}=y_i+1$, else $y_{i+1}=y_i$;
4. draw (x_{i+1}, y_{i+1});
5. calculate P_{i+1}:
   - if $P_i>0$ then $P_{i+1}=P_i+2\mathrm{d}y-2\mathrm{d}x$,
   - else $P_{i+1}=P_i+2\mathrm{d}y$;
6. $i=i+1$; if $i<\mathrm{d}x+1$ then goto 3; else end



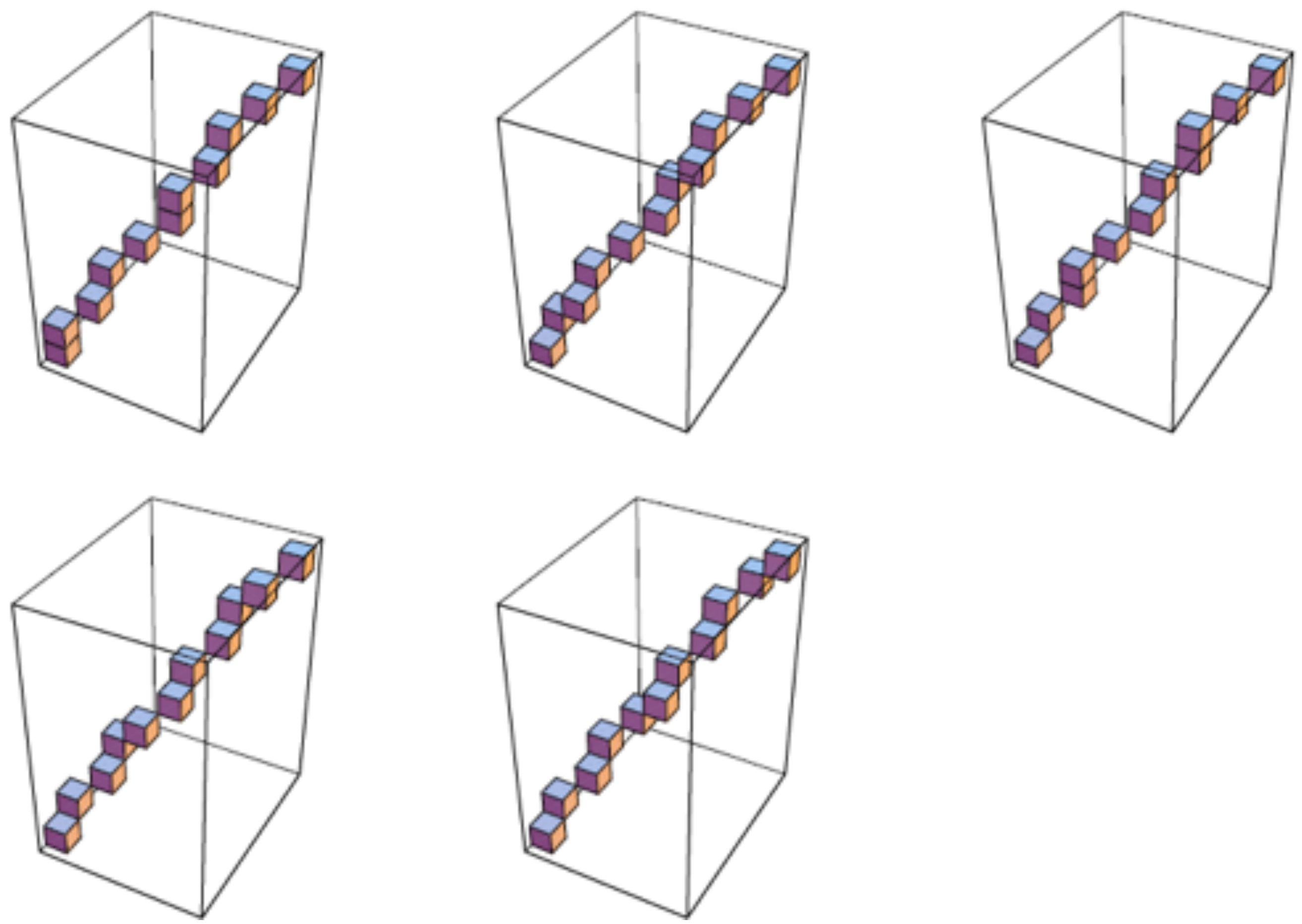
Question: Is it faster than DDA ?

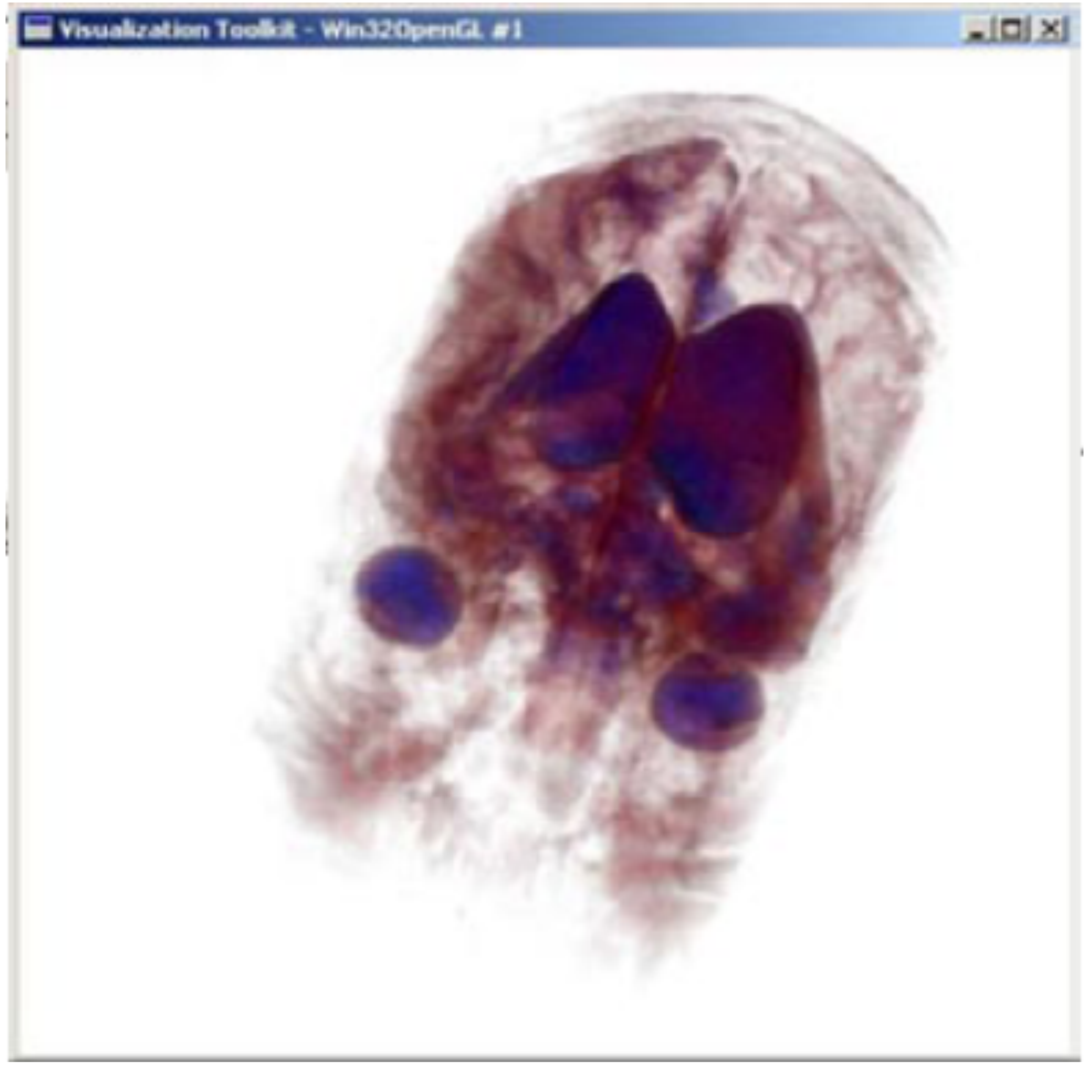
3D DDA and 3D Bresenham

Volume Rendering



3D DDA and 3D Bresenham algorithm





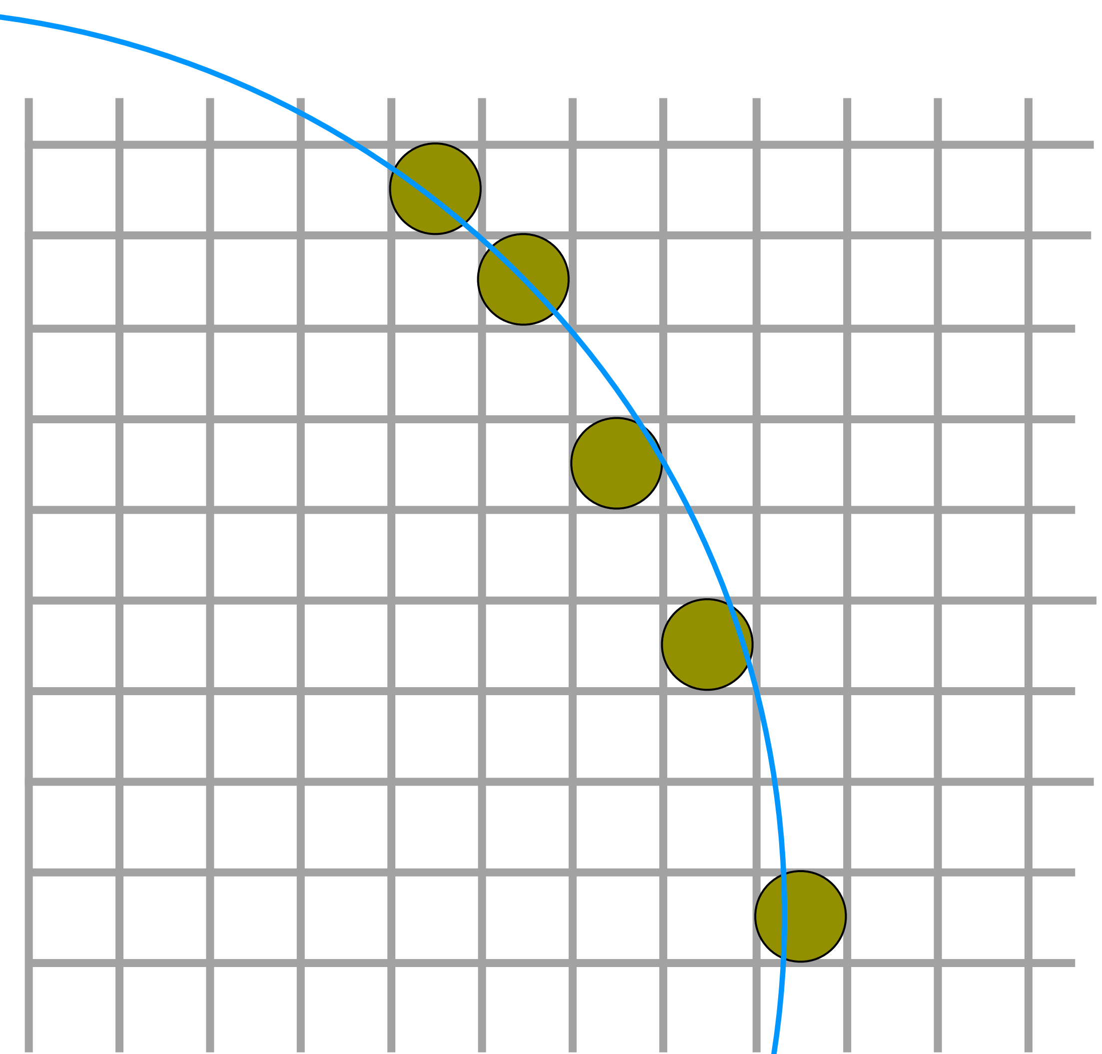
Scan converting circles

A circle with center (x_c, y_c) and radius r:

$$(x-x_c)^2 + (y-y_c)^2 = r^2$$

orthogonal coordinate

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$



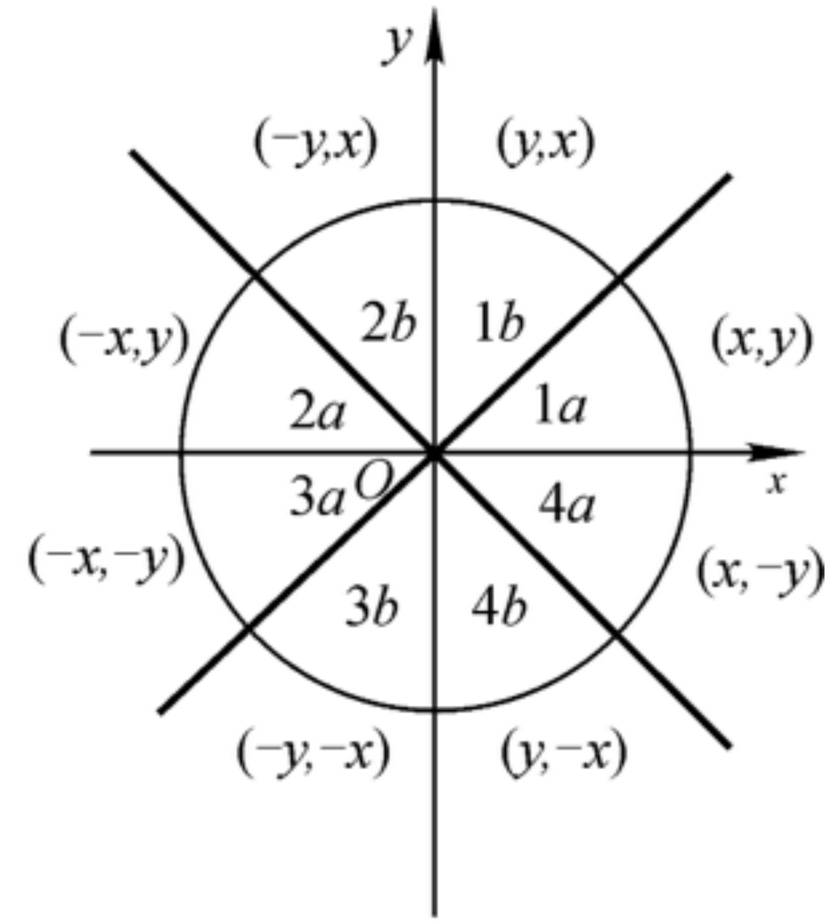
polar coordinates

$x = x_c + r \cdot \cos\theta$

$y = y_c + r \cdot \sin\theta$

$x_i = x_c + r \cdot \cos(i * \Delta\theta)$

$y_i = y_c + r \cdot \sin(i * \Delta\theta)$



Can be accelerated by symmetrical characteristic

$\theta = i * \Delta\theta, \quad i=0,1,2,3,\ldots.$

Discussion: How to speed up?

polar coordinates

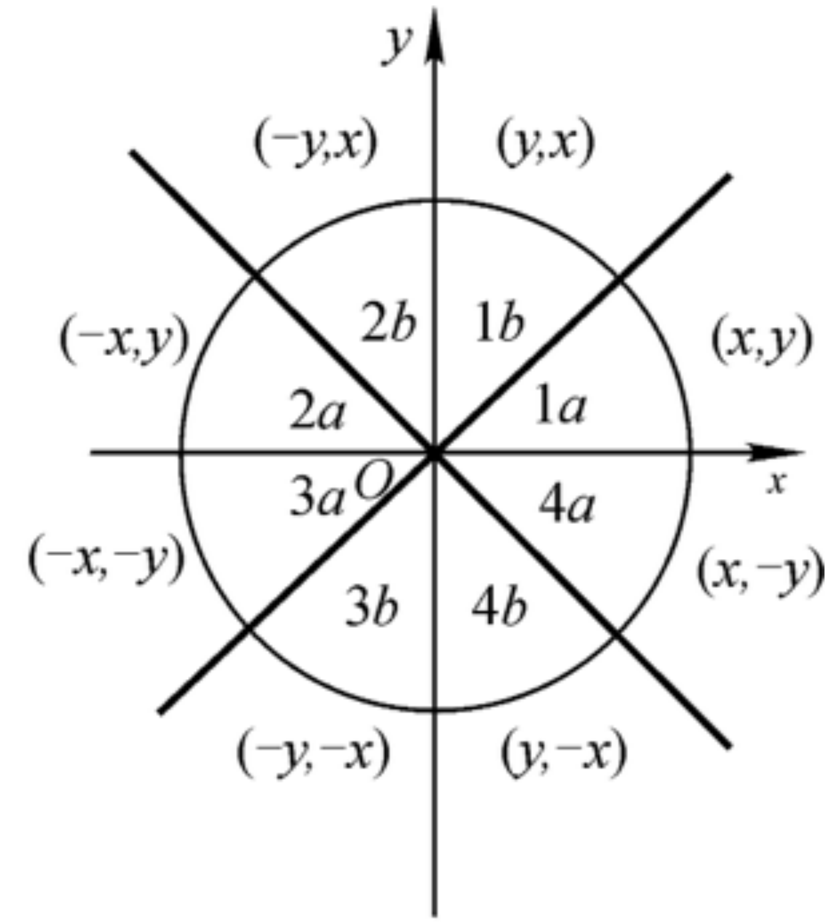
$x = x_c + r \cdot \cos\theta$

$y = y_c + r \cdot \sin\theta$

$x_i = x_c + r \cdot \cos(i * \Delta\theta)$

$y_i = y_c + r \cdot \sin(i * \Delta\theta)$

Can be accelerated by symmetrical characteristic



$\theta = i * \Delta\theta, \quad i=0,1,2,3,\ldots.$

Discussion: How to speed up?

$x_i = r \cos\theta_i$

$y_i = r \sin\theta_i$

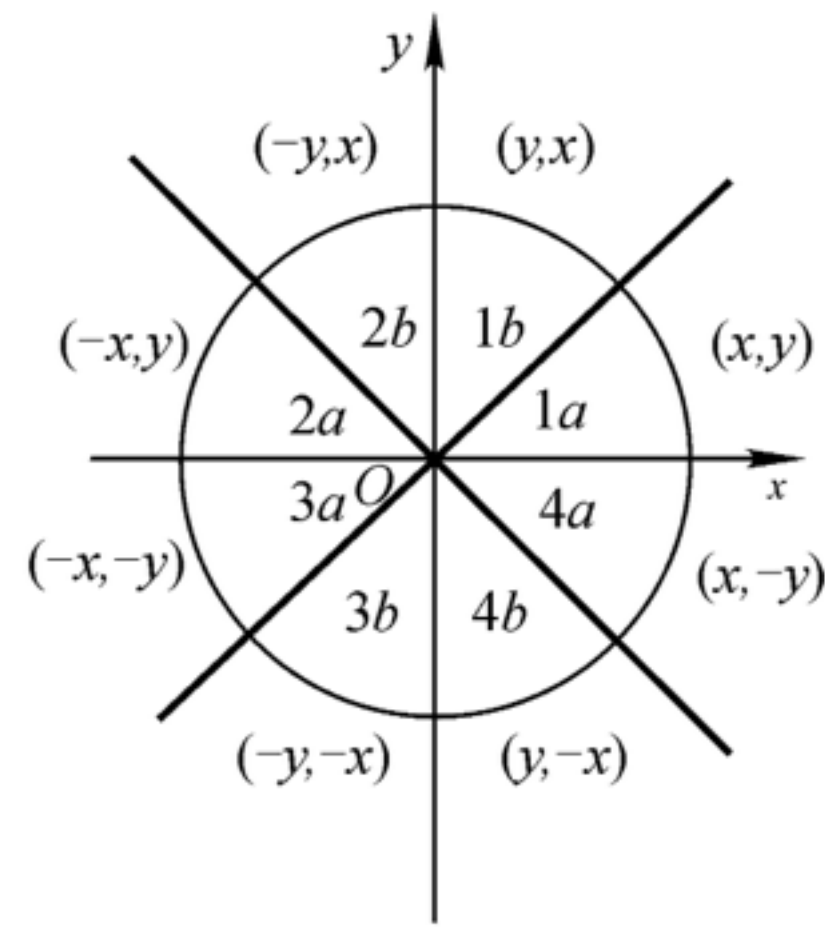
polar coordinates

$x = x_c + r \cdot \cos\theta$

$y = y_c + r \cdot \sin\theta$

$x_i = x_c + r \cdot \cos(i * \Delta\theta)$

$y_i = y_c + r \cdot \sin(i * \Delta\theta)$



Can be accelerated by symmetrical characteristic

$\theta = i * \Delta\theta, \quad i=0,1,2,3,\ldots.$

Discussion: How to speed up?

$x_i = r\cos\theta_i$

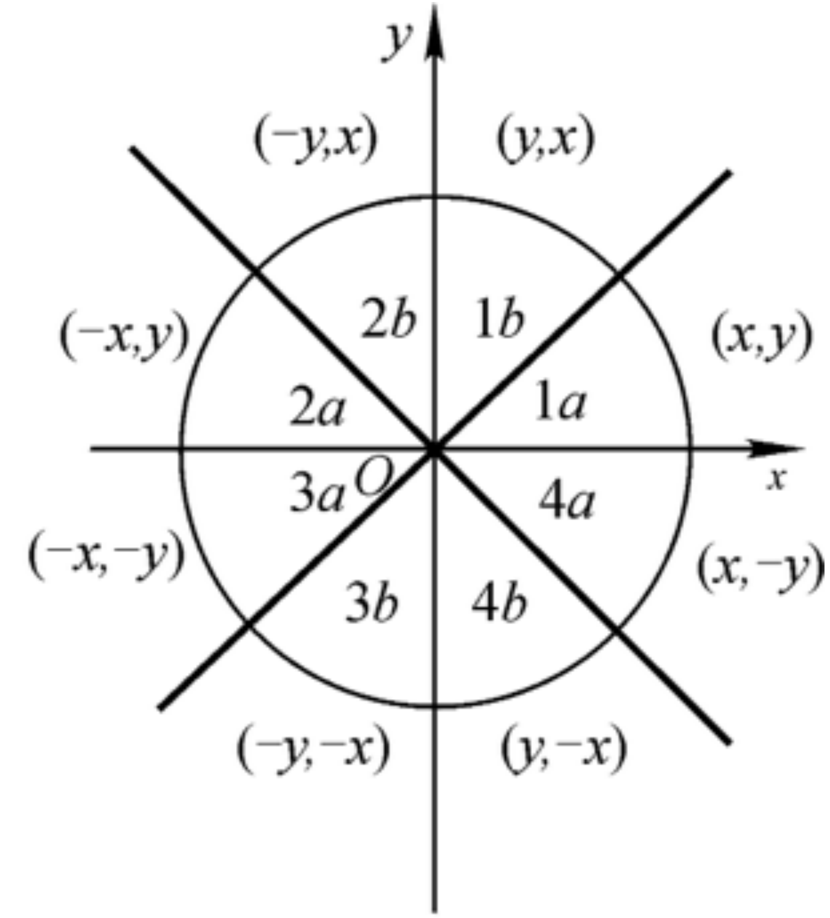
$y_i = r\sin\theta_i$

$$\begin{aligned} x_{i+1} &= r\cos(\theta_i + \Delta\theta) \\ &= r\cos\theta_i\cos\Delta\theta - r\sin\theta_i\sin\Delta\theta \\ &= x_i\cos\Delta\theta - y_i\sin\Delta\theta \end{aligned}$$

polar coordinates

$$x = x_c + r \cdot \cos\theta$$
$$y = y_c + r \cdot \sin\theta$$

$$x_i = x_c + r \cdot \cos(i * \Delta\theta)$$
$$y_i = y_c + r \cdot \sin(i * \Delta\theta)$$



Can be accelerated by symmetrical characteristic

$$\theta = i * \Delta\theta, \quad i=0,1,2,3,\ldots.$$

Discussion: How to speed up?

$$x_i = r\cos\theta_i$$
$$y_i = r\sin\theta_i$$

$$x_{i+1} = r\cos(\theta_i + \Delta\theta)$$
$$= r\cos\theta_i\cos\Delta\theta - r\sin\theta_i\sin\Delta\theta$$
$$= x_i\cos\Delta\theta - y_i\sin\Delta\theta$$

Bresenham Algorithm

Different representations

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2} \longrightarrow y = f(x) \quad x \in (x_0,\ x_1)$$

(explicit curve)

$$\begin{aligned} x &= x_c + r \cdot \cos\theta \\ y &= y_c + r \cdot \sin\theta \end{aligned} \longrightarrow \begin{cases} x = x(t) \\ y = y(t) \end{cases} \quad t \in (t_0,\ t_1)$$

(parametric curve)

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \longrightarrow g(x, y) = 0$$

(implicit curve)

Discussion: How to display an explicit curve,
How to display a parametric curve

Homework 1

- 提交
  - A4幅面（手写、word、pdf均可），注明姓名、学号
  - 10月10日前，提交电子文档给助教（可拍照）
- 详细给出一个完整的椭圆绘制（光栅化）算法
  - 输入：
    - 长短轴 a, b 整数
    - 圆心 (a, b)
  - 输出：
    - 在 大小为NxN 的frame buffer （2xa，2xb < N）中输出

  - 说明：方法的特点 (附加：如何绘制有线宽的椭圆)

3. OpenGL: A first look

What is OpenGL?

- “A software interface to graphics hardware”
- A graphics library (modeling and rendering)
- Very fast (a standard to be accelerated)
- Open standard
  - Was SGI’s IRIS GL
  - Regularly released by the Khronos Group
- OpenGL 1.0 (January, 1992)
- OpenGL 4.5 (August, 2014)

Before and with OpenGL

- Before:
  - IRIS GL
  - GKS
  - PHIGS / PHIGS+ (http://en.wikipedia.org/wiki/PHIGS)

- Other
  - VRML/X3D
  - Direct3D

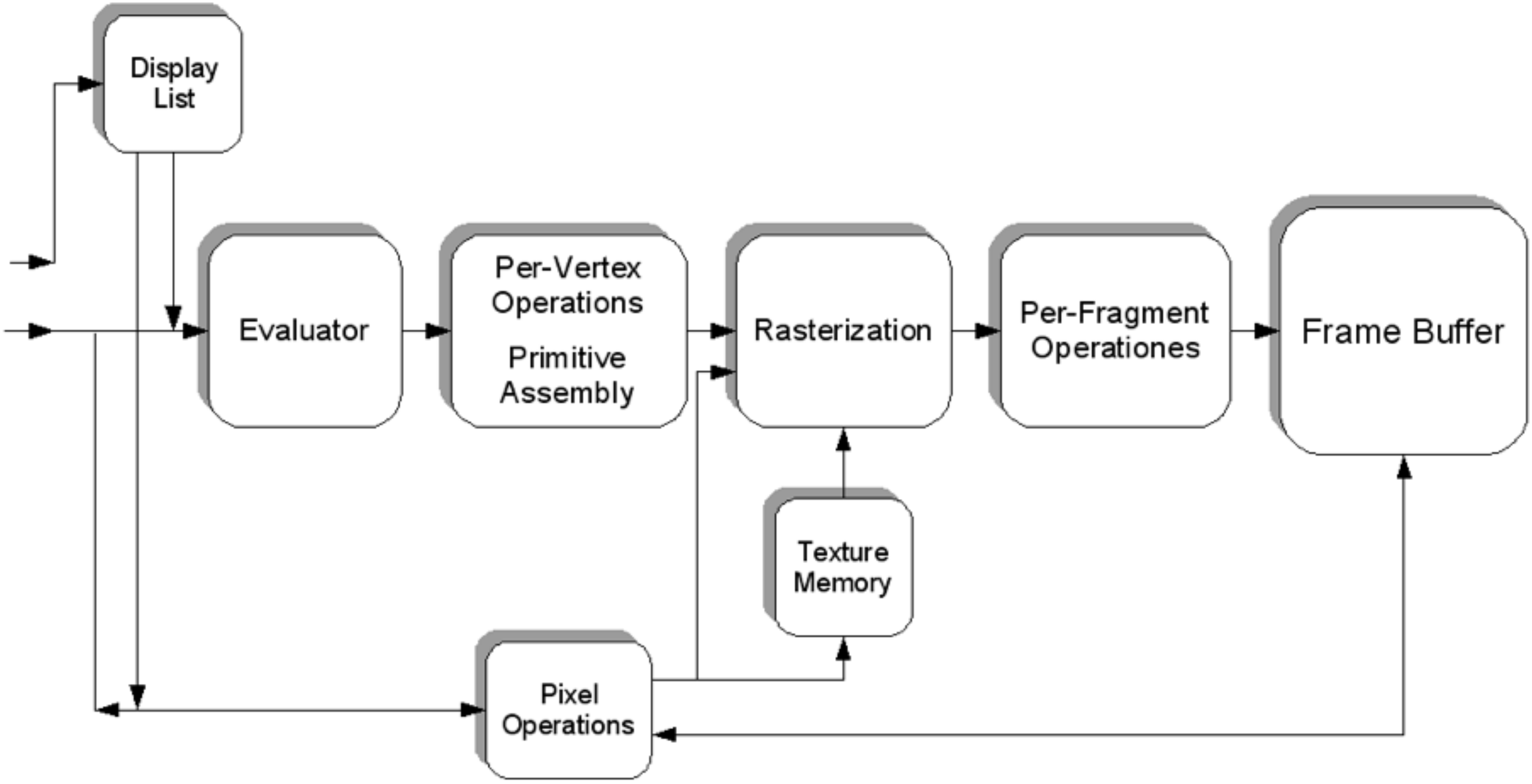
OpenGL today

- OpenGL 4.5 (August 11, 2014)
  - Direct State Access (DSA)
  - Flush Control
  - Robustness - providing a secure platform for applications such as WebGL browsers
  - OpenGL ES 3.1 API and shader compatibility
  - DX11 emulation features
- OpenGL NG, fusion with OpenGL ES

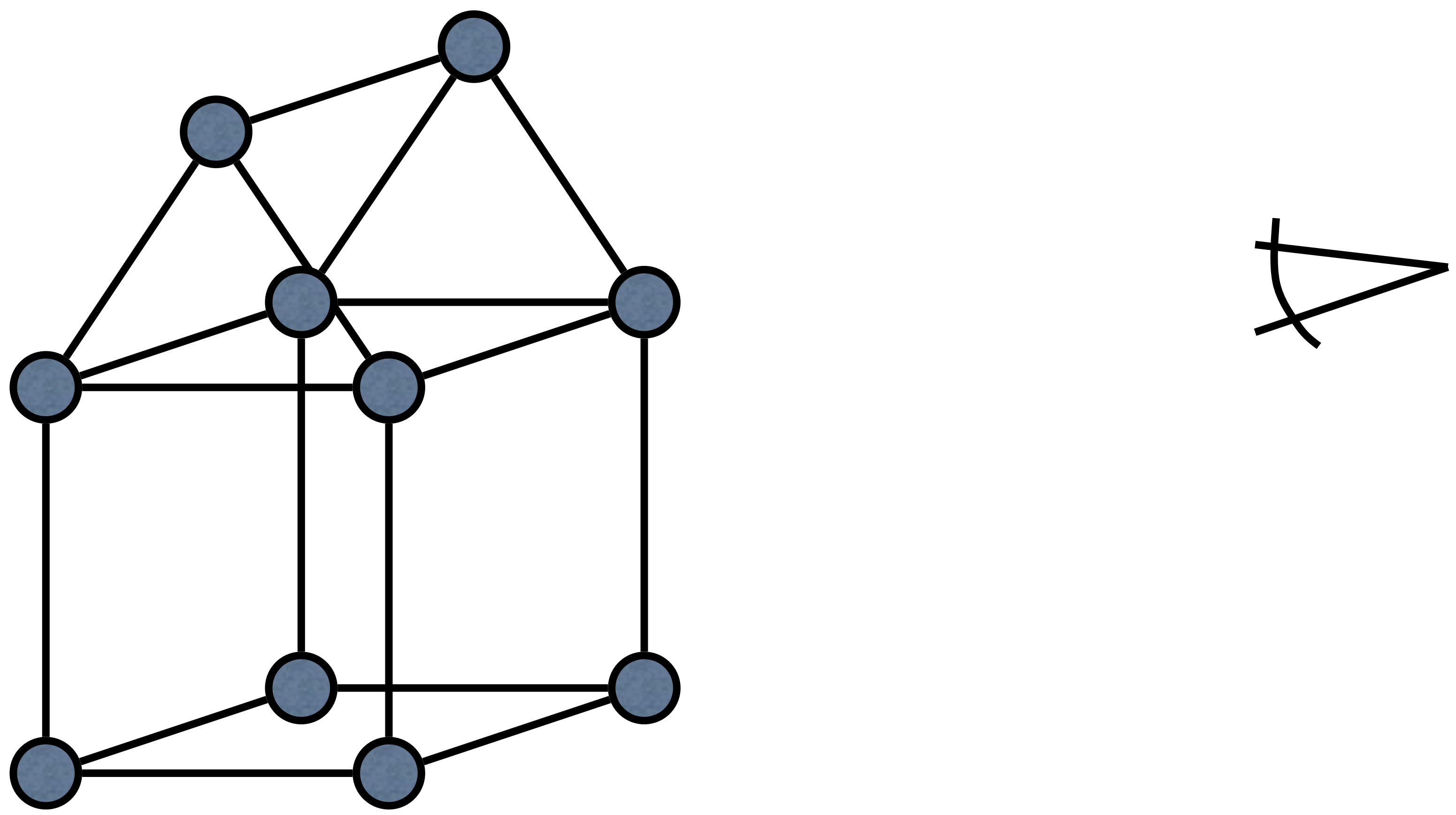
Create new world in OpenGL

- https://minecraft.net/

A Graphics Pipeline Process



Given 3D data, generate 2D view



OpenGL

- OpenGL is a multi-platform graphics API
- Applications make calls to OpenGL, which then renders an image (by handling the graphics hardware) and displays it
- The API contains about 150 commands
- Provides NO platform-dependent functionality (input, windowing, etc.)
- toolkit: GLUT

What OpenGL Does

- Allows definition of object shapes, material properties and lighting
- Arranges objects and interprets synthetic camera in 3D space
- Converts mathematical representations of objects into pixels (rasterization)
- Calculates the color of every object

OpenGL

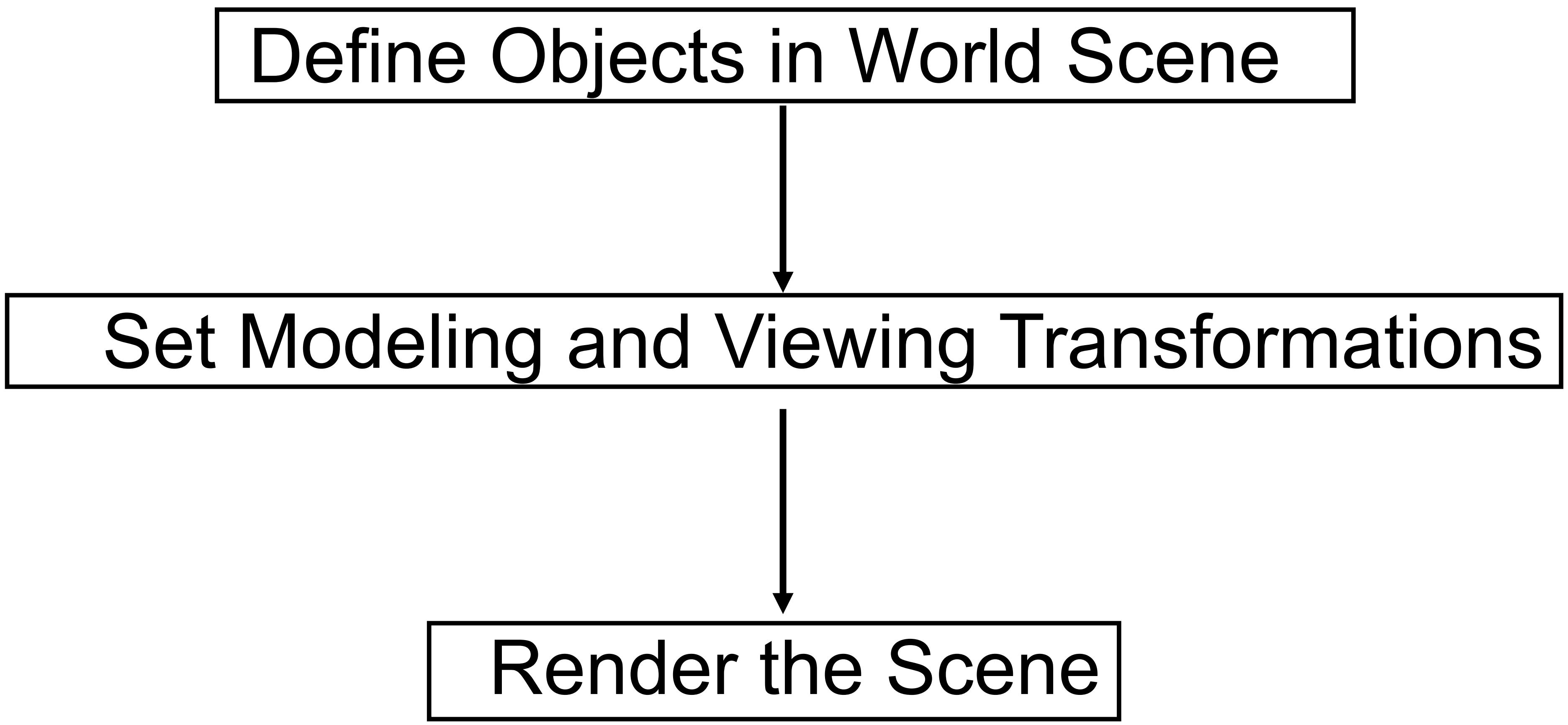
- NO high-level rendering functions for complex objects
  - build your shapes from primitives, points, lines, polygons, etc.
- The utility library GLU provides additional support

OpenGL tool chain

- OpenGL #include <GL/gl.h>
  - the "core" library that is platform independent
- GLU #include <GL/glu.h>
  - an auxiliary library that handles a variety of graphics accessory functions
- GLUT #include <GL/glut.h>
  - an auxiliary library that handles window creation, OS system calls (mouse buttons, movement, keyboard, etc), callbacks

- GLUI is a GUI manager written by Paul Rademacher (rademach@cs.unc.edu).

3 Stages in OpenGL



How OpenGL Works

- OpenGL is a state machine
  - You give it orders to set the current state of any one of its internal variables, or to query for its current status
  - The current state won't change until you specify otherwise
  - Ex.: if you set the current color to Red, everything you draw will be painted Red until you change the color explicitly
  - Each of the system's state variables has a default value

Example Code

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (
    GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

    glutInitWindowPosition(100,100);
    glutInitWindowSize(300,300);
    glutCreateWindow ("square");

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 10.0, 0.0, 10.0, -1.0, 1.0);

    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

```
void display(void)
{
    glClear( GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_POLYGON);
        glVertex3f(2.0, 4.0, 0.0);
        glVertex3f(8.0, 4.0, 0.0);
        glVertex3f(8.0, 6.0, 0.0);
        glVertex3f(2.0, 6.0, 0.0);
    glEnd();
    glFlush();
}
```

OpenGL Primitives

- GL_POINTS
- GL_LINES
- GL_LINE_STRIP
- GL_LINE_LOOP
- GL_TRIANGLES
- GL_QUADS
- GL_POLYGON
- GL_TRIANGLE_STRIP
- GL_TRIANGLE_FAN
- GL_QUAD_STRIP

1. GL_POLYGON and GL_TRIANGLE are the only ones in common usage
2. valid OpenGL polygons are closed, convex, co-planar and non-intersecting, which is always true for triangles!

Examples

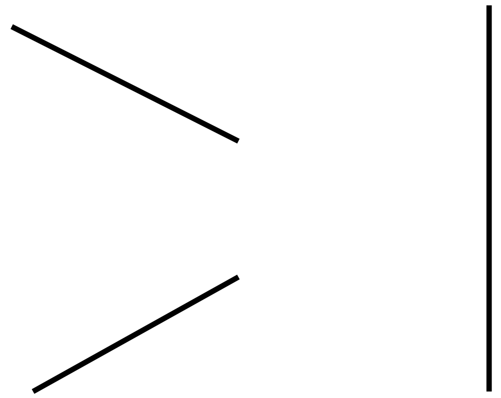
```
glBegin(GL_POLYGON);
        glVertex2i(0,0);
        glVertex2i(0,1);
        glVertex2i(1,1);
        glVertex2i(1,0);
    glEnd() ;
```

```
glBegin(GL_POINTS);
        glVertex2i(0,0);
        glVertex2i(0,1);
        glVertex2i(1,1);
        glVertex2i(1,0);
    glEnd() ;
```

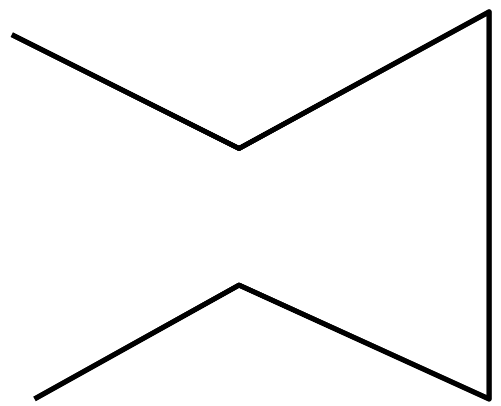
Examples

```
GLfloat list[6][2] ;
```

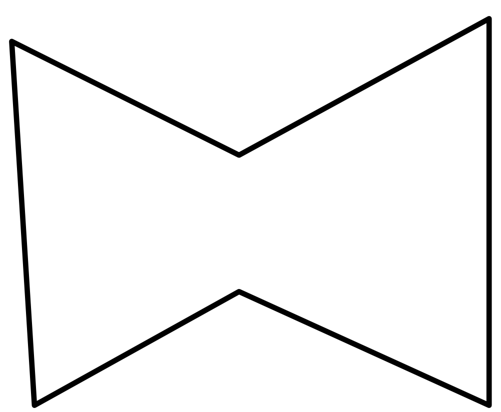
```
glBegin(GL_LINES)
for (int i = 0 ; i < 6 ;i++)
    glVertex2v(list[i]);
glEnd() ;
```



```
glBegin(GL_LINE_STRIP)
for (int i = 0 ; i < 6 ;i++)
    glVertex2v(list[i]);
glEnd() ;
```



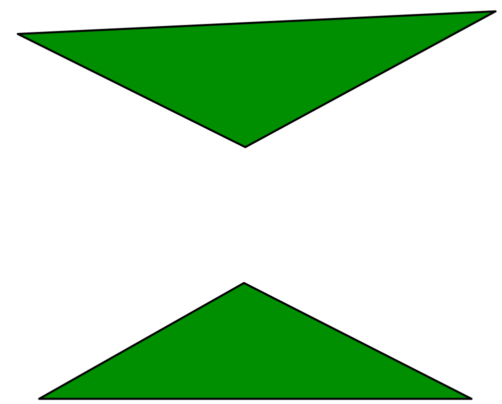
```
glBegin(GL_LINE_LOOP)
for (int i = 0 ; i < 6 ;i++)
    glVertex2v(list[i]);
glEnd() ;
```



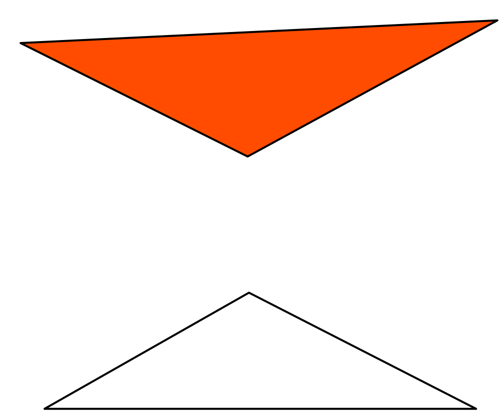
Examples

```
GLfloat list[6][2] ;

glColor3f(0.0, 1.0, 0.0);
glBegin(GL_TRIANGLES)
    for (int i = 0 ; i < 6 ;i++)
        glVertex2v(list[i]);
glEnd() ;
```

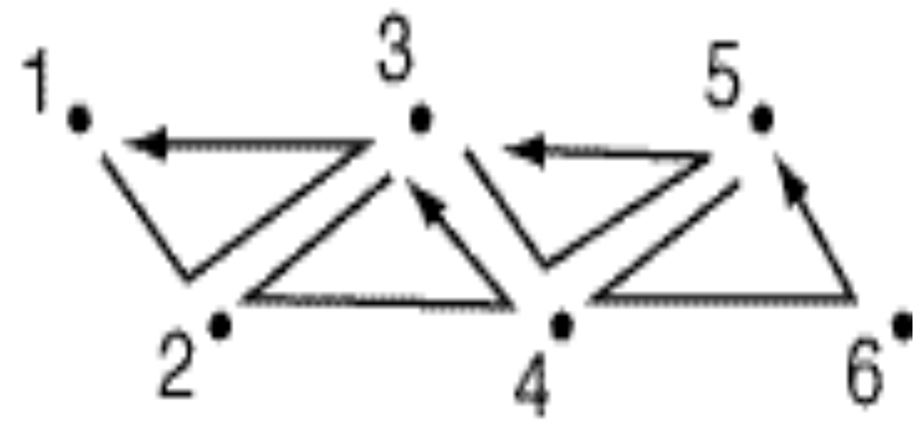


```
glBegin(GL_TRIANGLES)
    glColor3f(1.0, 0.0, 0.0);
    for ( i = 0 ; i < 3 ;i++)
        glVertex2v(list[i]);
    glColor3f(1.0, 1.0, 1.0);
    for ( i = 3 ; i < 6 ;i++)
        glVertex2v(list[i]);
glEnd() ;
```

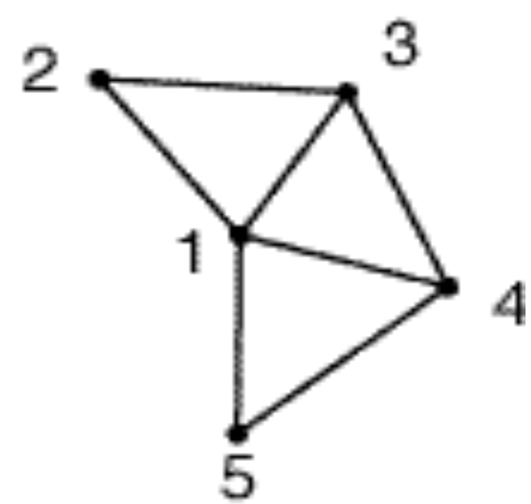


Examples

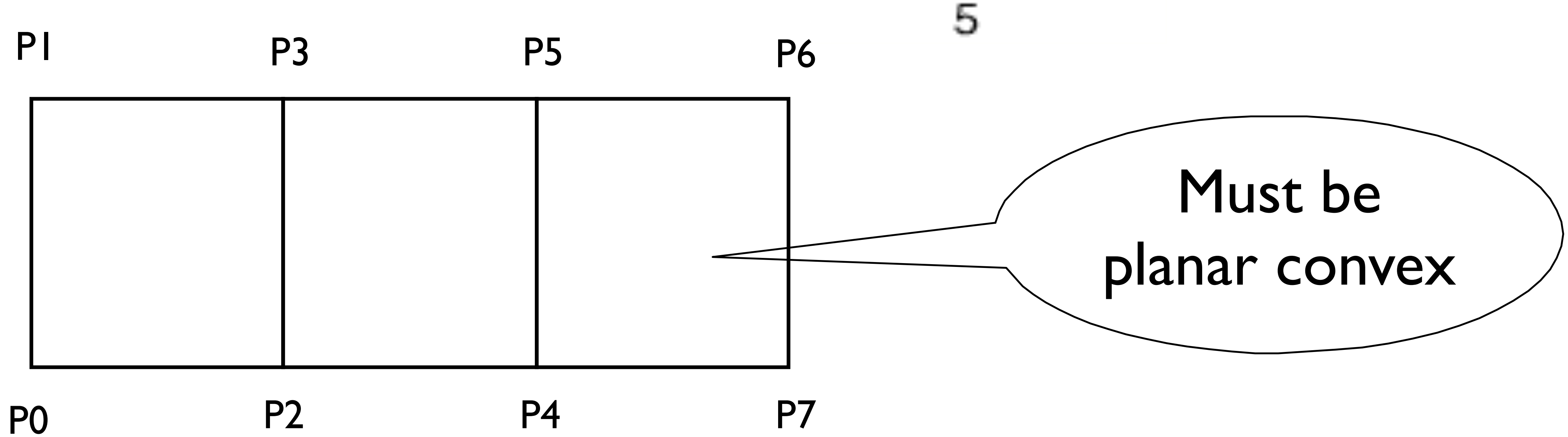
GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN



GL_QUAD_STRIP



OpenGL Command Syntax

- All command names begin with gl
  - Ex.: glVertex3f(0.0, 1.0, 1.0);
- Constant names are in all uppercase
  - Ex.: GL_COLOR_BUFFER_BIT
- Data types begin with GL
  - Ex.: GLfloat onevertex[3];
- Most commands end in two characters that determine the data type of expected arguments
  - Ex.: glVertex3f(…) => 3 GLfloat arguments

glVertex

- All primitives are defined in terms of vertices
  - glVertex2f(x, y);
  - glVertex3f(x, y, z);
  - glVertex4f(x, y, z, w);
  - glVertex3fv(a);   // with a[0], a[1], a[2]

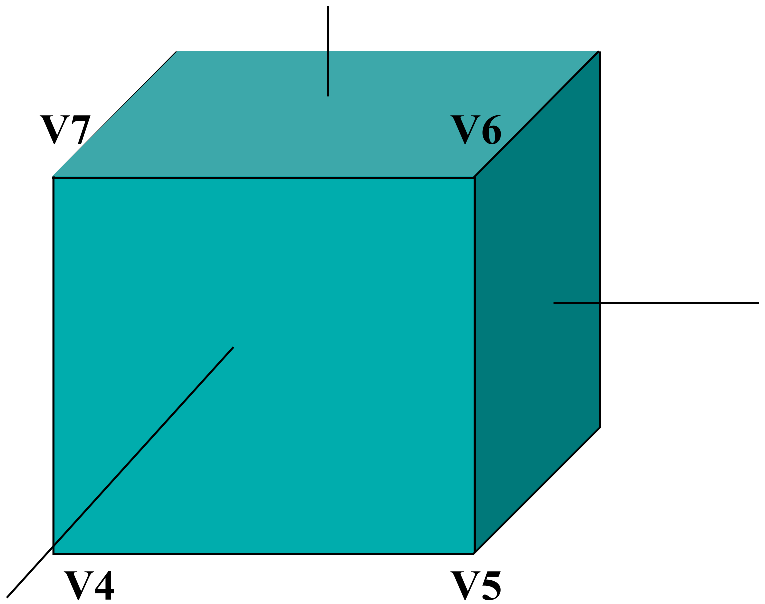
Building Objects From Vertices

- Specify a primitive mode, and enclose a set of vertices in a glBegin / glEnd block
- glBegin(GL_POLYGON);
- glVertex3f(1.0, 2.0, 0.0);
- glVertex3f(0.0, 0.0, 0.0);
- glVertex3f(3.0, 0.0, 0.0);
- glVertex3f(3.0, 2.0, 0.0);
- glEnd();

OpenGL Example

```
void drawOneCubeface(size)
{
  static Glfloat v[8][3];
   v[0][0] = v[3][0] = v[4][0] = v[7][0] = -size/2.0;
   v[1][0] = v[2][0] = v[5][0] = v[6][0] =  size/2.0;
   v[0][1] = v[1][1] = v[4][1] = v[5][1] = -size/2.0;
   v[2][1] = v[3][1] = v[6][1] = v[7][1] =  size/2.0;
   v[0][2] = v[1][2] = v[2][2] = v[3][2] = -size/2.0;
   v[4][2] = v[5][2] = v[6][2] = v[7][2] =  size/2.0;

  glBegin(GL_POLYGON);
    glVertex3fv(v[0]);
    glVertex3fv(v[1]);
    glVertex3fv(v[2]);
    glVertex3fv(v[3]);
  glEnd();
}
```



Colors

- OpenGL colors are typically defined as RGB components
  - each of which is a float in the range [0.0, 1.0]

- For the screen's background:
  - glClearColor(0.0, 0.0, 0.0); // black color
  - glClear(GL_COLOR_BUFFER_BIT);
- For objects:
  - glColor3f(1.0, 1.0, 1.0); // white color

Other Commands in glBegin / glEnd blocks

- Not every OpenGL command can be located in such a block. Those that can include, among others:
  - glColor
  - glNormal (to define a normal vector)
  - glTexCoord (to define texture coordinates)
  - glMaterial (to set material properties)

Example

```
glBegin( GL_POLYGON );
    glColor3f( 1.0, 1.0, 0.0 ); glVertex3f( 0.0, 0.0, 0.0 );
    glColor3f( 0.0, 1.0, 1.0 ); glVertex3f( 5.0, 0.0, 0.0 );
    glColor3f( 1.0, 0.0, 1.0 ); glVertex3f( 0.0, 5.0, 0.0 );
glEnd();
```

Polygon Display Modes

- glPolygonMode(GLenum face, GLenum mode);
  - Faces: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
  - Modes: GL_FILL, GL_LINE, GL_POINT
  - By default, both the front and back face are drawn filled
- glFrontFace(GLenum mode);
  - Mode is either GL_CCW (default) or GL_CW
- glCullFace(Glenum mode);
  - Mode is either GL_FRONT, GL_BACK, GL_FRONT_AND_BACK;
- You must enable and disable culling with
  - glEnable(GL_CULL_FACE) or glDisable(GL_CULL_FACE);

Drawing Other Objects

- GLU contains calls to draw cylinders, cones and more complex surfaces called NURBS
- GLUT contains calls to draw spheres and cubes

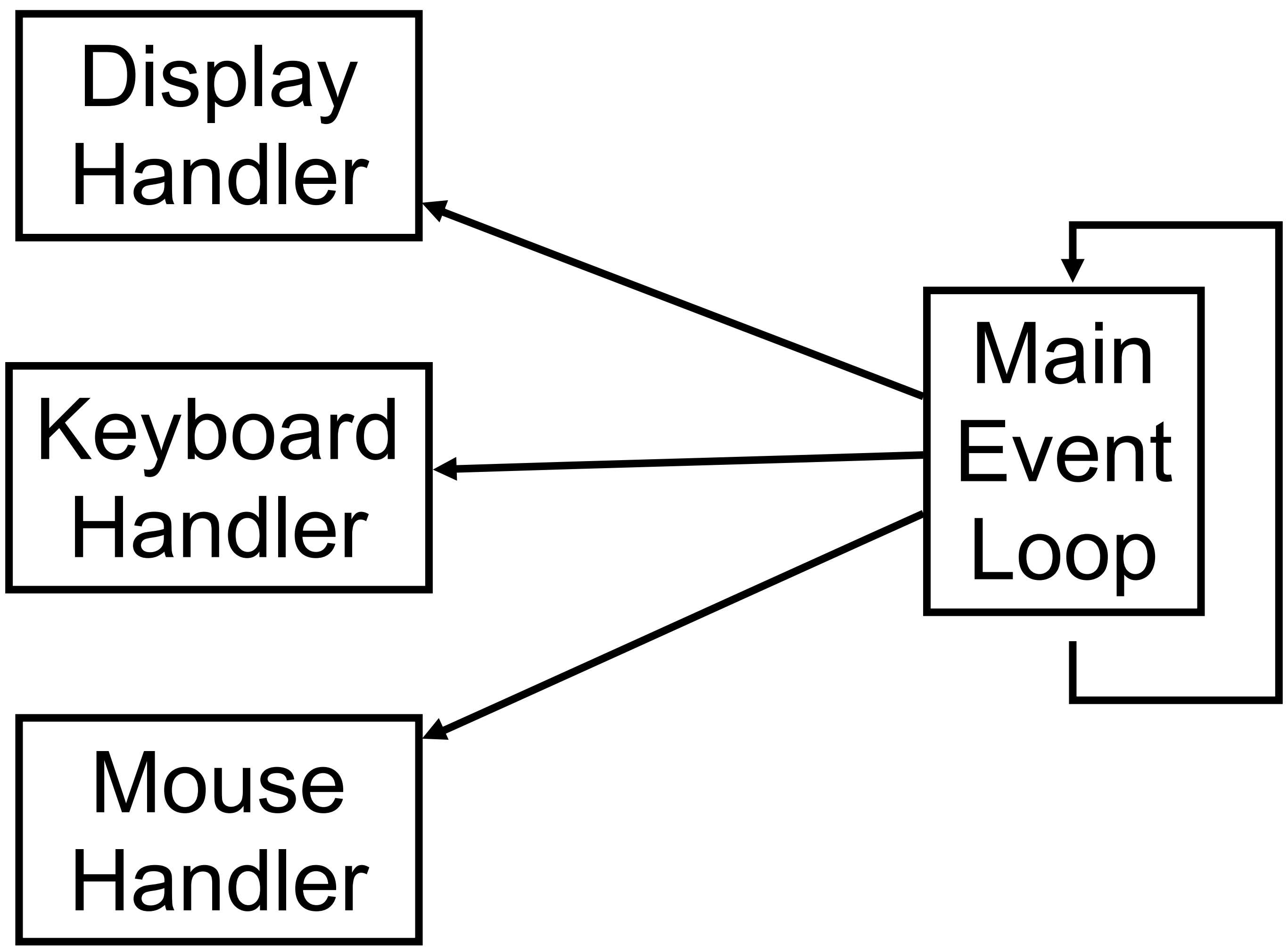
Compiling OpenGL Programs

- To use GLUT :
  - #include <GL/glut.h>
  - This takes care of every other include you need
  - Make sure that glut.lib (or glut32.lib) is in your compiler's library directory, and that the object module or DLL is also available
- See *OpenGL Game Programming* or online tutorials for details

Structure of GLUT-Assisted Programs

- GLUT relies on user-defined callback functions, which it calls whenever some event occurs
  - Function to display the screen
  - Function to resize the viewport
  - Functions to handle keyboard and mouse events

Event Driven Programming



Simple GLUT Example

Displaying a square

```
int main (int argc,  char *argv[])
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGBA | LUT_DOUBLE);

   int windowHandle
       = glutCreateWindow("Simple GLUT App");

   glutDisplayFunc(redraw);
   glutMainLoop();

   return 0;
}
```

Display Callback

Called when window is redrawn

```
void redraw()
{
  glClear(GL_COLOR_BUFFER_BIT);

  glBegin(GL_QUADS);
  glColor3f(1, 0, 0);
    glVertex3f(-0.5,  0.5, 0.5);
    glVertex3f( 0.5,  0.5, 0.5);
    glVertex3f( 0.5, -0.5, 0.5);
    glVertex3f(-0.5, -0.5, 0.5);
  glEnd(); // GL_QUADS

  glutSwapBuffers();
}
```

More GLUT

Additional GLUT functions

```
glutPositionWindow(int x,int y);
glutReshapeWindow(int w, int h);
```

Additional callback functions

```
glutReshapeFunction(reshape);
glutMouseFunction(mousebutton);
glutMotionFunction(motion);
glutKeyboardFunction(keyboardCB);
glutSpecialFunction(special);
glutIdleFunction(animate);
```

Reshape Callback

Called when the window is resized

```
void reshape(int w, int h)
{
   glViewport(0.0,0.0,w,h);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0,w,0.0,h, -1.0, 1.0);

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

Mouse Callbacks

Called when the mouse button is pressed

```
void mousebutton(int button, int state, int x, int y)
{
   if (button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
   {
      rx = x; ry = winHeight – y;
   }
}
```

Called when the mouse is moved with button down

```
void motion(int x, int y)
{
   rx = x; ry = winHeight – y;
}
```

Keyboard Callbacks

Called when a button is pressed

```
void keyboardCB(unsigned char key, int x, int y)
{
  switch(key)
  { case 'a': cout<<"a Pressed"<<endl; break; }
}
```

Called when a special button is pressed

```
void special(int key, int x, int y)
{
  switch(key)
  { case GLUT_F1_KEY:
      cout<<"F1 Pressed"<<endl; break; }
}
```

OpenGL – GLUT Example

```
#include <gl/glut.h>
#include <stdlib.h>
static GLfloat spin = 0.0;
void init( void )
{
  glClearColor( 0.0, 0.0, 0.0, 0.0 );
  glShadeModel( GL_FLAT );
}
```

```
void display( void )
{
glClear( GL_COLOR_BUFFER_BIT );
glPushMatrix();
glRotatef( spin, 0.0, 0.0, 1.0 );
glColor3f( 1.0, 1.0, 1.0 );
glRectf( -25.0, -25.0, 25.0, 25.0 );
glPopMatrix();
glutSwapBuffers();
}
```

OpenGL – GLUT Example

```
void spinDisplay( void )
{
    spin += 2.0;
    if( spin > 360.0 )
    spin -= 360.0;
    glutPostRedisplay();
}
```

```
void reshape( int w, int h )
{
    glViewport( 0, 0, (GLsizei) w, (GLsizei)
    h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho( -50.0, 50.0, -50.0, 50.0, -1.0, 1.0 );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
}
```

OpenGL – GLUT Example

```
void mouse( int button, int state, int x, int y )
{
    switch( button )
    {
     case GLUT_LEFT_BUTTON:
        if( state == GLUT_DOWN )
            glutIdleFunc( spinDisplay );
        break;
     case GLUT_RIGHT_BUTTON:
        if( state == GLUT_DOWN )
            glutIdleFunc( NULL );
        break;
     default:    break;
    }
}
```

OpenGL – GLUT Example

```
int main( int argc, char ** argv )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGB );
    glutInitWindowSize( 250, 250 );
    glutInitWindowPosition( 100, 100 );
    glutCreateWindow( argv[ 0 ] );

    init();
    glutDisplayFunc( display );
    glutReshapeFunc( reshape );
    glutMouseFunc( mouse );
    glutMainLoop();
    return 0;
}
```

Web Resources

http://www.opengl.org

http://nehe.gamedev.net

http://www.xmission.com/~nate/glut.html

祝国庆假期愉快！