

Introduction to Computer Graphics

Farhana Bandukwala, PhD

Lecture 5: OpenGL objects and
transformations

Outline

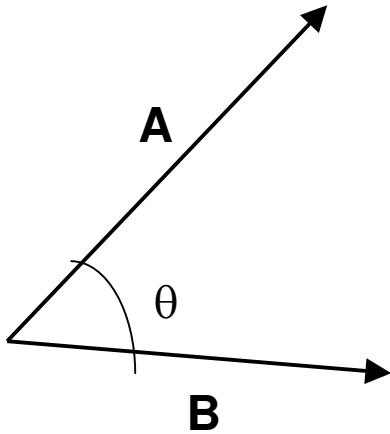
- Geometric primitives
- World vs Screen Coordinates
- World to viewport Transformation
- Transformation matrices in Open GL

OpenGL 2D Examples



**All rendered using Animo, a
2D rendering software**

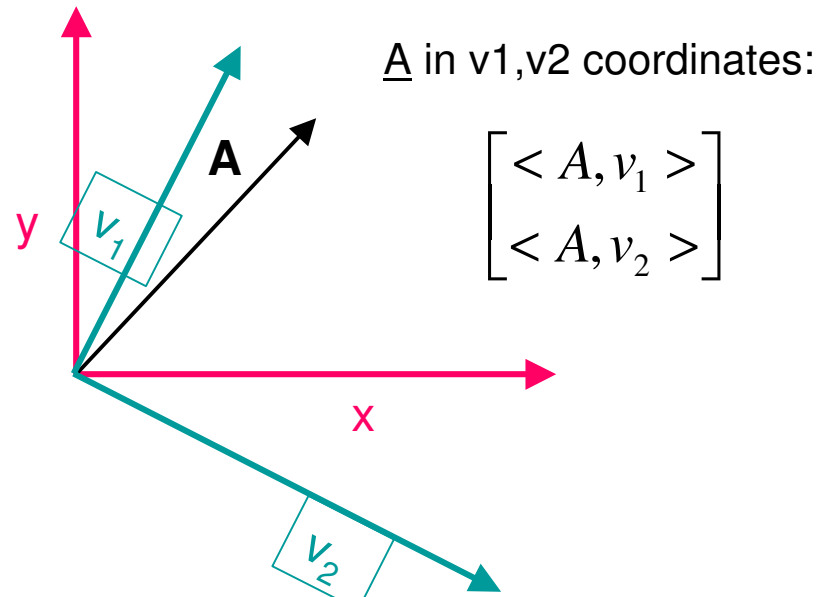
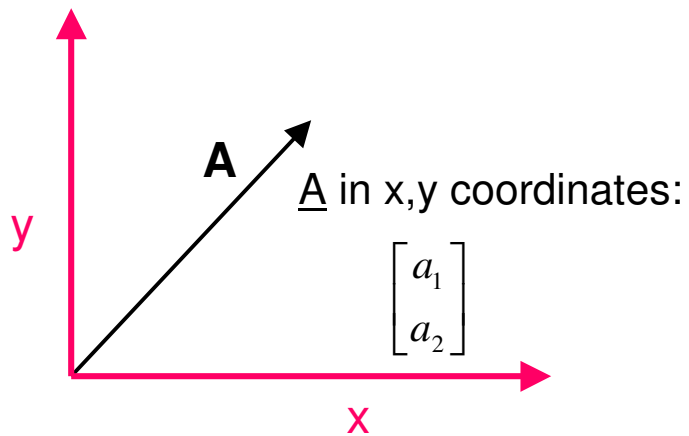
Dot Product (revisited)



$$\langle A, B \rangle = |A| |B| \cos \theta$$

$$\cos \theta = \frac{[a_1, a_2] * [b_1, b_2]^T}{\sqrt{[a_1, a_2] * [a_1, a_2]^T} * \sqrt{[b_1, b_2] * [b_1, b_2]^T}}$$

Represents projection of one vector onto the other



Geometric Primitives

- Points = vertex `glVertex2i(GLint x, GLint y)` or `glVertex2f(GLfloat x, GLfloat y)`
- Finite vectors: lines defined by 2 vertices
- Polygon: closed primitive composed of n vertices
- Assuring convexity: triangles

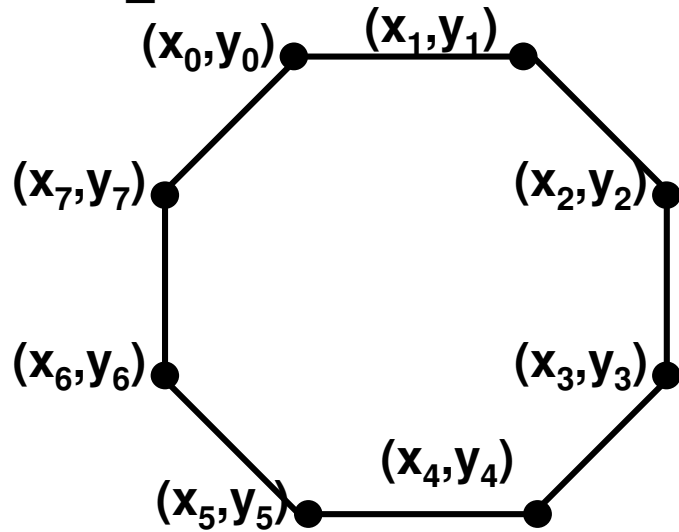
```
glBegin(GL_POINTS)
glVertex2f(x1,y1);
glVertex2f(x2,y2);
...
glEnd()
```

```
glBegin(GL_LINES)
glVertex2f(x1,y1);
glVertex2f(x2,y2);
...
glEnd()
```

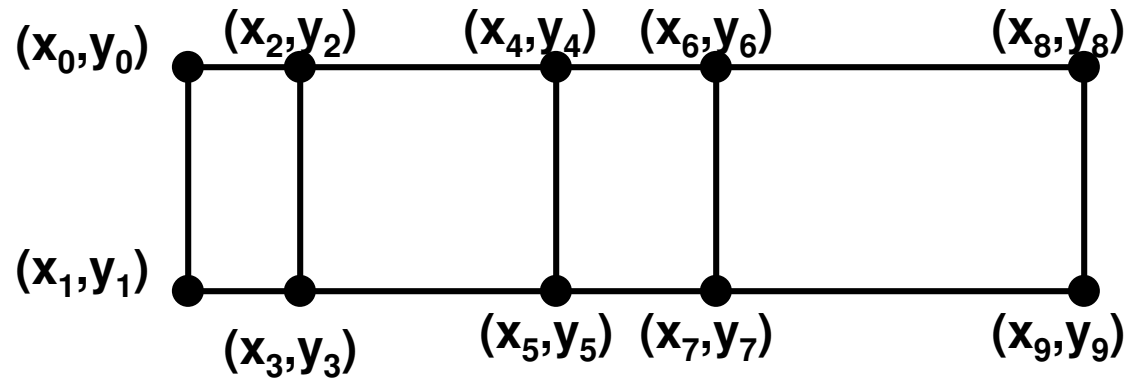
- **GL_LINE_STRIP**
- **GL_LINE_LOOP**
- **GL_POLYGON**

Geometric Primitives (contd.)

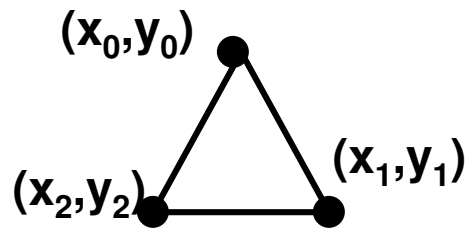
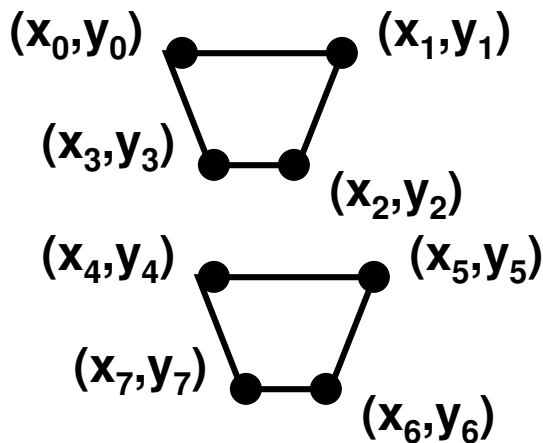
GL_POLYGON



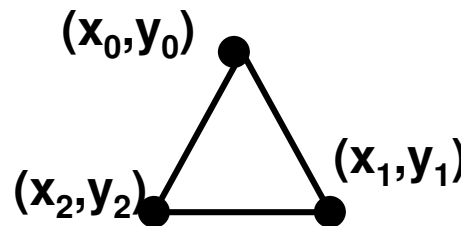
GL_QUAD_STRIP



GL_QUADS



GL_TRIANGLES



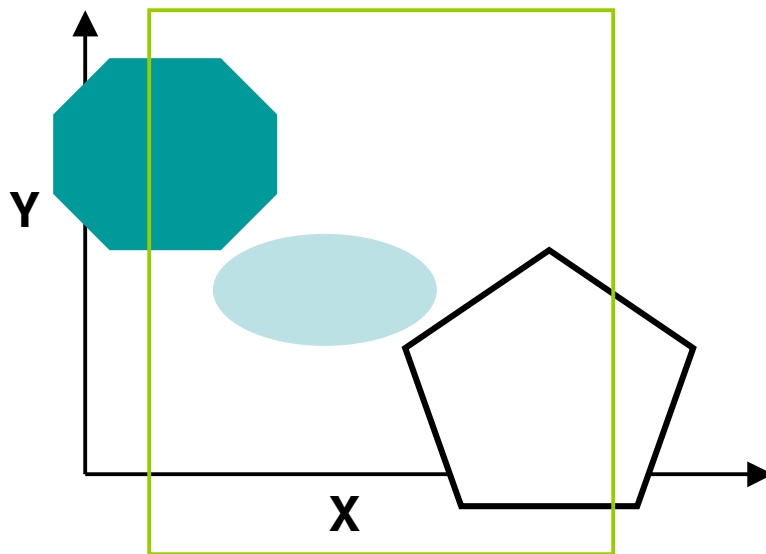
- GL_TRIANGLE_STRIP
- GL_TRIANGLE_FAN

Types of polygons

- Simple polygon: no crossed edges
- Convex: all points between 2 interior points also interior (triangles, rectangles)
- Concave polygons acceptable but not optimal
- Edges only (mesh)
- Solid, shaded, texture-mapped (`glPolygonMode()`)

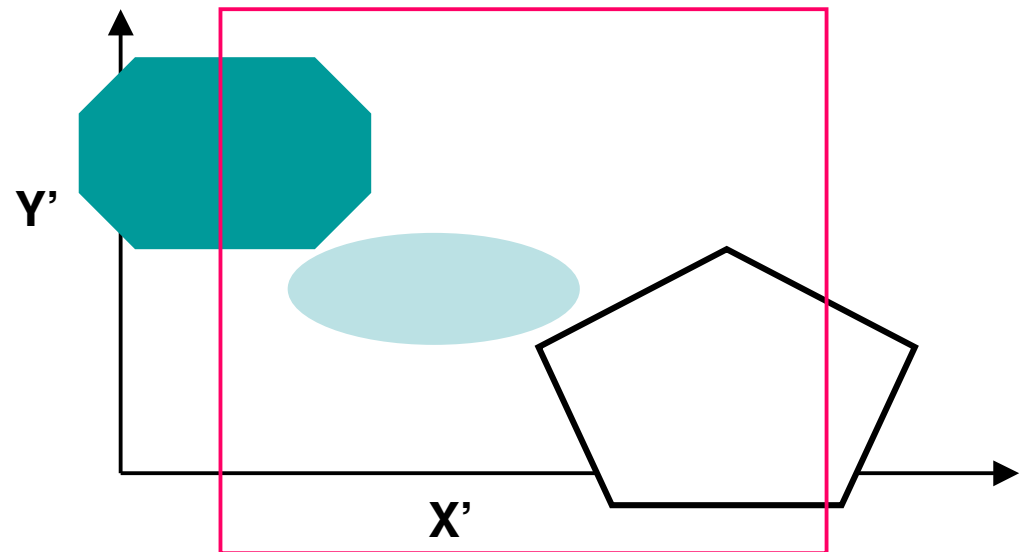
Viewing vs Screen coordinate system

Two dimensional viewing world
in physical world dimensions



`gluOrtho2D(GLdouble left,
GLdouble right,
GLdouble bottom,
GLdouble top)`

Viewing rectangle in pixels
(screen coordinates)



`glViewport(GLdouble left,
GLdouble bottom,
GLdouble width,
GLdouble height)`

Transformations: world to viewport

- Need to explicitly specify which matrix gets transformation in Open GL
- Projection matrix:
 - to transform from world coordinates to viewing coordinates
 - `glMatrixMode(GL_PROJECTION)`
- Model-view matrix:
 - to position model within viewing volume
 - duality between model & camera placement
 - `glMatrixMode(GL_MODELVIEW)`

Transformation tips

- Choose one convention and stick with it
- Option 1:
 - normalize viewing volume: `gluOrtho2d(0.0,1.0,0.0,1.0)`
 - Transform objects in model to fit in viewing volume:
`glMatrixMode(GL_MODELVIEW), glRotate2f(),
glTranslate2f(),glScale2f()`
- Option 2:
 - Transform viewing volume: `gluOrtho2d(xmin,xmax,ymin,ymax),
glMatrixMode(GL_PROJECTION)`
 - Fix objects in world

Putting it all together

Initialize window and rendering parameters

...

Before each display:

```
glViewport(0,0,GLint width, GLint height);
```

```
glMatrixMode(GL_PROJECTION);
```

```
glLoadIdentity();
```

```
gluOrtho2D(GLdouble left, GLdouble right,  
           GLdouble bottom, GLdouble top);
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity();
```

Optional transformations applied to align model in scene

```
glRotate2f();
```

```
glTranslate2f();
```

...

Specify primitives in world coordinates

...