Computer Graphics

Drawing Tools

Coordinate Systems

- The space in which objects are described is called **world coordinates**
  – the numbers used for x and y are those in the world, where the objects are defined
- World coordinates use the Cartesian *xy*-coordinate system used in mathematics
  – Based on whatever units are convenient.

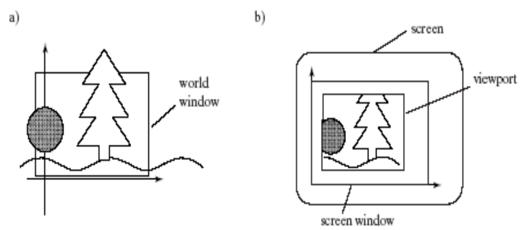
Coordinate Systems

- We define a rectangular **world window** in these world coordinates.
- The world window specifies which part of the world should be drawn:
  – whichever part lies inside the window should be drawn
  – whichever part lies outside should be clipped away and not drawn.
- OpenGL does the clipping automatically.

Coordinate Systems

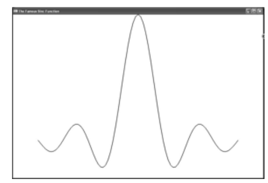
- We also define a rectangular **viewport** in the screen window on the display.
  – Pixels x pixels
- A mapping between the world window and the viewport is established by OpenGL.
  – consisting of scalings [change size] and translations [move object]
- The objects inside the world window appear automatically at proper sizes and locations inside the viewport
  – in **screen coordinates,** which are pixel coordinates on the display

Windows and Viewport



Coordinate Systems Example

- We want to graph $\mathrm{sinc}(x) = \dfrac{\sin(\pi x)}{\pi x}$
- Sinc(0) = 1 by definition. Interesting parts of the function are in $-4.0 \le x \le 4.0$.
- What should be the world coordinates?
- What should be the viewport?



Coordinate Systems Example

- The program which graphs this function is given in Fig. 3.3.
- The function setWindow sets the world window size:

```
void setWindow(GLdouble left, GLdouble right, GLdouble bottom,
    GLdouble top)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(left, right, bottom, top);
}
```


Coordinate Systems Example

- The function setViewport sets the screen viewport size and location

```
void setViewport(GLint left, GLint right, GLint bottom, GLint top)
{
  glViewport(left, bottom, right - left, top - bottom);
}
```

- Calls: setWindow(-5.0, 5.0, -0.3, 1.0);
  setViewport(0, 640, 0, 480);

Coordinate Systems Example

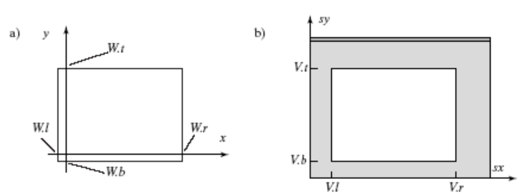
- Modeling objects is done in the world coordinates
  – glBegin(GL_LINE_STRIP); etc
- This allows us to focus on modeling instead of projections, clipping, etc.
- These ideas extend to 3D, but we will worry about that later…

Windows and Viewports

- We use natural coordinates for what we are drawing, the scene
- We define the world window to specify the part of the scene to display on screen
- OpenGL converts our coordinates to screen coordinates when we set up a screen window and a viewport.
- The viewport may be smaller than the screen window.
  – The default viewport is the entire screen window.
- The conversion requires scaling and shifting: mapping the world window to the screen window and the viewport.

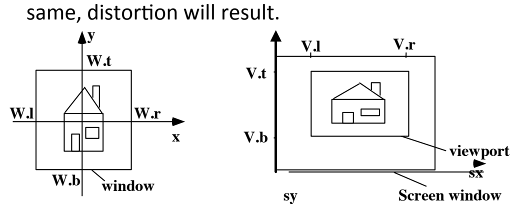
Mapping Windows

- Windows are described by their left, right, bottom, and top values, w.l, w.r, w.b, and w.t in world coordinates.
- Viewports are described by the same values: v.l, v.r, v.b, and v.t values but in screen window coordinates.



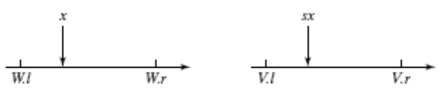
Mapping

- We can map any aligned rectangle to any other aligned rectangle.
  – If the aspect ratios of the 2 rectangles are not the same, distortion will result.



Window-to-Viewport Mapping

- We want our mapping to be proportional:
- For example,
  - if x is ¼ of the way between the left and right world window boundaries
  - then the screen x (s_x) should be ¼ of the way between the left and right viewport boundaries.



Middle Tennessee State University

Window-to-Viewport Mapping

- This requirement forces our mapping to be linear

$$sx = Ax + C$$
$$sy = By + D$$

  - We require $\dfrac{s_x - V.l}{V.r - V.l} = \dfrac{x - W.l}{W.r - W.l}$
  - How to solve for A, B, C, and D?
  - This allows us to map ANY point from world coordinates to viewport coordinates

Middle Tennessee State University

Window-to-Viewport Mapping

- Find the values for A, B, C, and D for the given setup
  - World Window (-10,10,-6,6)
  - Viewport (0,600,0,400)
    (left, right, bottom, top)
- Is aspect ratio preserved?

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

- What are the corresponding coordinates (sx, sy) for a point (5, 2) in the world window?

Middle Tennessee State University

GL Functions To Create the Map

- World window:

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

- Viewport:

  void glViewport(GLint x, GLint y, GLint width, GLint height);

  This sets the lower left corner of the viewport, along with its width and height.

Middle Tennessee State University

GL Functions To Create the Map

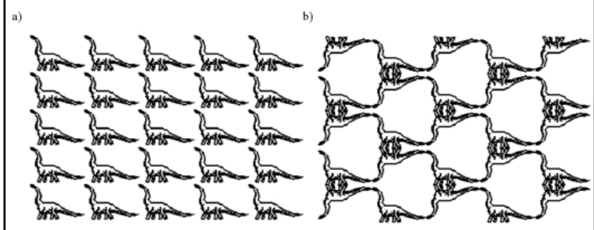
- Because OpenGL uses matrices to set up all its transformations, the call to gluOrtho2D() must be preceded by two setup functions:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

gluOrtho2D(-10, 10, -6, 6);
glViewport(0, 0, 600, 400);
```


Application:Tiling with Viewports

How would you do this?



Applications (continued)

- Tiling A was set up by the following code:

```
setWindow(0, 640.0, 0, 440.0);  // set a fixed window

for (int i = 0; i < 5; i++)        // for each column
   for (int j = 0; j < 5; j++)
   {       // for each row
           glViewport (i*64, j*44,64, 44);
           // set the next viewport
           drawPolylineFile("dino.dat"); // draw it again
   }
```


Applications (continued)

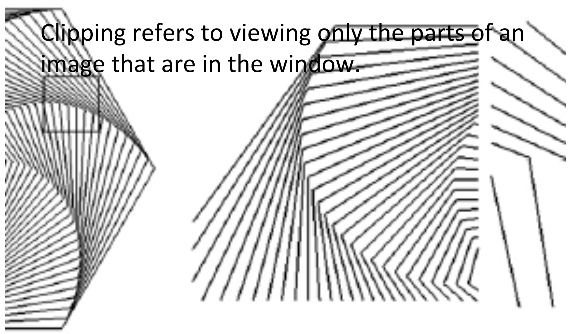
- Tiling B requires more effort: you can only turn a world window upside down, not a viewport.

```
for (int i = 0; i < 5; i++) // for each column
   for (int j = 0; j < 5; j++)
   {          // for each row
    if ((i + j) % 2 == 0)
        setWindow(0.0, 640.0, 0.0, 440.0);
    else
        setWindow(0.0, 640.0, 440.0, 0.0); // upside-down

    glViewport (i*64, j*44,64, 44);
        // no distortion
    drawPolylineFile("dino.dat");
   }
```

Application: Clip, Zoom and Pan

Clipping refers to viewing only the parts of an image that are in the window.

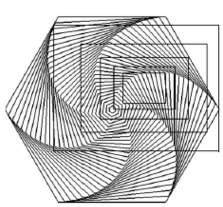


Application (continued)

- The figure is a collection of concentric hexagons of various sizes, each rotated slightly with respect to the previous one. It is drawn by a function called hexSwirl ();
- The figure showed 2 choices of world windows.
- We can also use world windows for zooming and roaming (panning).
- How would you change the windows to zoom?
- How would you change the windows to pan?

Zooming and Panning

- To zoom, we pick a concentric set of windows of decreasing size and display them from outside in.



Zooming and Roaming

- The animation of the zoom will probably not be very smooth.
- We want to look at one drawing while the next one is drawn
  - then switch to the new drawing.
- We use glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  - gives us 2 buffers, one to look at and one to draw in
- We add glutSwapBuffers(); after the call to hexSwirl (); // change to the new drawing

Roaming (Panning)

- To roam, or pan, we move a viewport through various portions of the world. This is easily accomplished by translating the window to a new position.
- How would you ensure the effect looked the same on different computers?

Resizing the Screen Window

- Users are free to alter the size and aspect ratio of the screen window.
- You may want GL to handle this event so that your drawing does not get distorted.
- Register the reshape function:

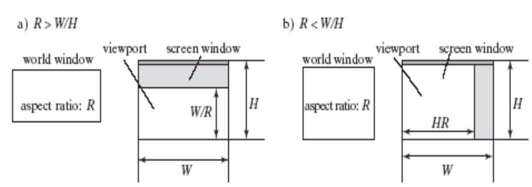
  glutReshapeFunc (myReshape);
- void myReshape (GLsizei W, GLsizei H);

  collects the new width and height for the window.
  - Should you modify the window or viewport?

Preserving Aspect Ratio

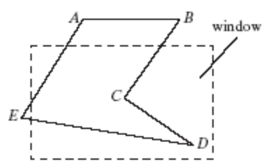
- We want the largest viewport which preserves the aspect ratio R of the world window.
- Suppose the screen window has width W and height H:
  - If R > W/H, the viewport should be width W and height W/R
  - If R < W/H, the viewport should be width H*R and height H
  - What happens if R = W/H?

Automatic Aspect Ratio Preservation for Viewports



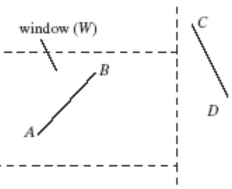
Clipping Lines

- We want to draw only the parts of lines that are inside the world window.
- To do this, we need to replace line portions outside the window by lines along the window boundaries.
- The process is called clipping the lines.



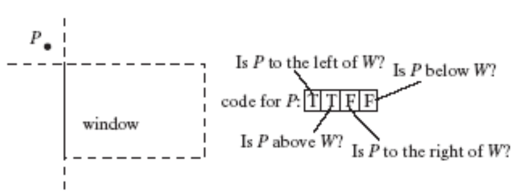
Clipping

- The method we will use is called Cohen-Sutherland clipping.
- There are 2 trivial cases:
  - a line AB totally inside the window, which we draw all of
  - a line CD totally outside the window, which we do not draw at all.



Clipping

- For all lines, we give each endpoint of the line a code specifying where it lies relative to the window W:



Clipping

- The diagram below shows Boolean codes for the 9 possible regions the endpoint lies in (left, above, below, right).

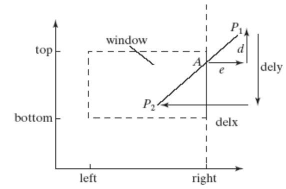
TTFF	FTFF	FTTF
TFFF	FFFF window	FFTF
TFFT	FFFT	FFTT

Clipping 4 cases

- P1 and P2 are both FFFF
  - Trivial Accept
- P1 and P2 have a T in the same element
  - Trivial Reject
- One point is FFFF and one has at least one T
  - Clipping required
- Other cases
  - Clipping may be required

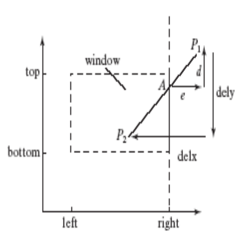
Clipping

- A line consists of 2 endpoints (vertices),
  - P_1 and P_2.
- If we do not have a trivial case, we must alter the vertices to the points where the line intersects the window boundaries (replace P_1 by A).



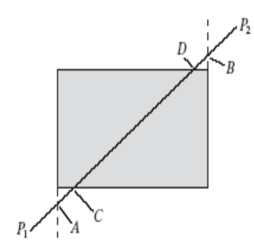
Clipping

- In the diagram, d/dely = e/delx (similar triangles).
- Also,
  - delx = p_1.x – p_2.x,
  - dely = p_1.y – p_2.y and
  - e = p_1.x – W.r.
- Obviously, A.x = W.r.
- A.y=?



A Line Needing 4 Clips

- The intersection A
  - x = W.l
- The intersection B
  - x = W.r
- The intersection C
  - y =W.b
- The intersection D
  - y =W.t



Cohen Sutherland Algorithm

- Determine if the line needs clipping
  - Compare TFTT of P1 and P2
- If it does, choose which of the 4 sides to clip against
  - Determine which of the two points need clipping
  - Determine which side does clipping occur
  - Test all 4 sides to make sure the line is fully clipped

Clipping by hand…

- Given a window (50,120,0,100)
- Go through the clipping algorithm for the following line segments
  - P1 (50,40), p2 (100,20)
  - P1 (10,170), p2 (100,0)
  - P1 (20,-10), p2 (200,200)
- Give the new endpoints for any clipped segments

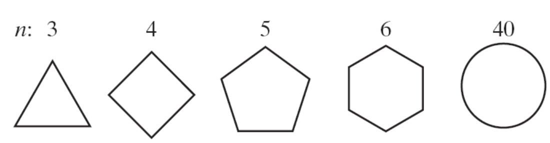
Middle Tennessee State University

Drawing Regular Polygons, Circles, and Arcs

- A polygon is **simple** if no two of its edges cross each other. More precisely, only adjacent edges can touch and only at their shared endpoint.
- A polygon is **regular** if it is simple, if all its sides have equal length, and if adjacent sides meet at equal interior angles.
- We give the name ***n*-gon** to a regular polygon having *n* sides; familiar examples are the 4-gon (a square), an 8-gon (a regular octagon) and so on.

Middle Tennessee State University

Regular Polygons



Middle Tennessee State University

Drawing Circles and Arcs

- Two methods:
  - **The center is given, along with a point on the circle.**
    - Here drawCircle(IntPoint center, int radius) can be used as soon as the radius is known.
    - If *c* is the center and *p* is the given point on the circle, the radius is simply the distance from *c* to *p*, found using the usual Pythagorean Theorem.

Drawing Circles and Arcs

- **Three points are given through which the circle must pass.**
  - It is known that a unique circle passes through any three points that don't lie in a straight line.
  - Finding the center and radius of this circle is discussed in Chapter 4.

Parametric Curves

- Three forms of equation for a given curve:
  - Explicit: e.g., y = m*x + b
  - Implicit: F(x, y) = 0; e.g., y – m*x –b = 0
  - Parametric: x = x(t), y = y(t)

  t is a parameter; frequently, $0 \le t \le 1$.

  Example: Line:

  F(x, y) = $A_x + (B_x - A_x) * t$
    - A and B are 2D points with x and y values. A and B are on the straight line
    - The parametric form is

      $x(t) = A_x + (B_x - A_x)*t$

      $y(t) = A_y + (B_y - A_y)*t$

Specific Parametric Forms

- line:

  $x(t) = A_x + (B_x - A_x)*t$, t>=0

  $y(t) = A_y + (B_y - A_y)*t$
- circle:

  x(t) = r*cos(t), 0 <= t <=2π

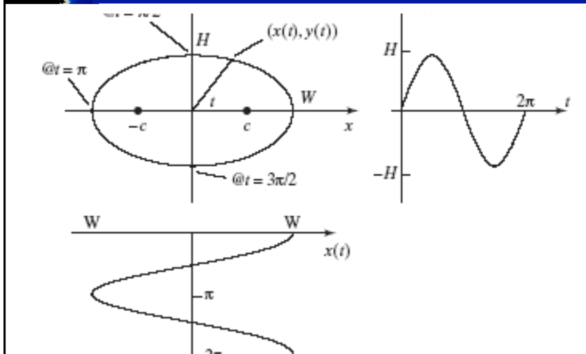
  y(t) = r*sin(t)
- ellipse:

  x(t) = W*cos(t), 0 <= t <=2π

  y(t) = H*sin(t)

  – W and H are half-width and half-height.

Example for Ellipse



Finding Implicit Form from Parametric Form

- Combine the x(t) and y(t) equations to eliminate t.
- Example:

  ellipse:

  x = W*cos(t)

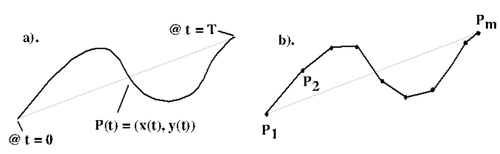
  y = H*sin(t)

  – $x^2 = W^2\cos^2(t)$, $y^2 = H^2\sin^2(t)$

  – Dividing by the W or H factors and adding gives $(x/W)^2 + (y/H)^2 = 1$, the implicit form.

Drawing Parametric Curves

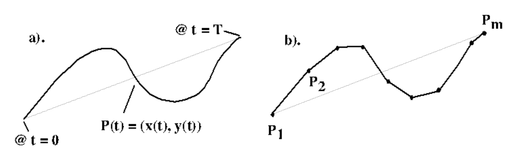
- For a curve *C* with the parametric form *P*(*t*) = (*x*(*t*), *y*(*t*)) as *t* varies from 0 to *T*, we use **samples** of *P*(*t*) at closely spaced instants.



Drawing Parametric Curves

– The position $P_i = P(t_i) = (x(t_i), y(t_i))$ is calculated for a sequence $\{t_i\}$ of times.

– The curve *P*(*t*) is approximated by the polyline based on this sequence of points P_i.



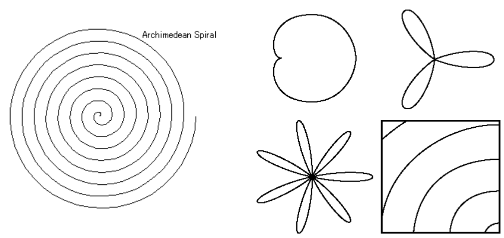
Drawing Parametric Curves

- Code:

```
// draw the curve (x(t), y(t)) using
// the array t[0],..,t[n-1] of sample times
glBegin(GL_LINES);
  for(int i = 0; i < n; i++)
    glVertex2f((x(t[i]), y(t[i]));
glEnd();
```


Shapes as Parametric Curves

- For drawing purposes, parametric forms circumvent all of the difficulties of implicit and explicit forms.
- Cardioid, 2 rose curves, Archimedean spiral



Polar Coordinates Parametric Form

- General form: $x = f(\theta)*\cos(\theta)$
  $y = f(\theta)*\sin\theta$
- cardioid: $f(\theta) = K*(1 + \cos(\theta))$, $0 \le \theta \le 2\pi$
- rose: $f(\theta) = K* \cos(n*\theta)$, $0 \le \theta \le 2n\pi$, where n is number of petals (n odd) or twice the number of petals (n even)
- spirals: Archimedean, $f(\theta) = K\theta$
  Logarithmic, $f(\theta) = Ke^{a\theta}$
- K is a scale factor for the curves.

Polar coordinates Parametric Form

– conic sections (ellipse, hyperbola, circle, parabola): $f(\theta) = (1 \pm e \cos \theta)^{-1}$
  - e is eccentricity:
    1 : parabola;
    0 : circle;
    between 0 and 1, ellipse;
    greater than 1, hyperbola