Points, Circles & Lines

OpenGL

Learning Outcomes

- Understand the role of primitives & vertices in Opengl
- Be able to draw points and circles
- Understand how the point size it acquired and set
- Be able to draw individual lines, connected lines, line strips and loops
- Have seen line stippling

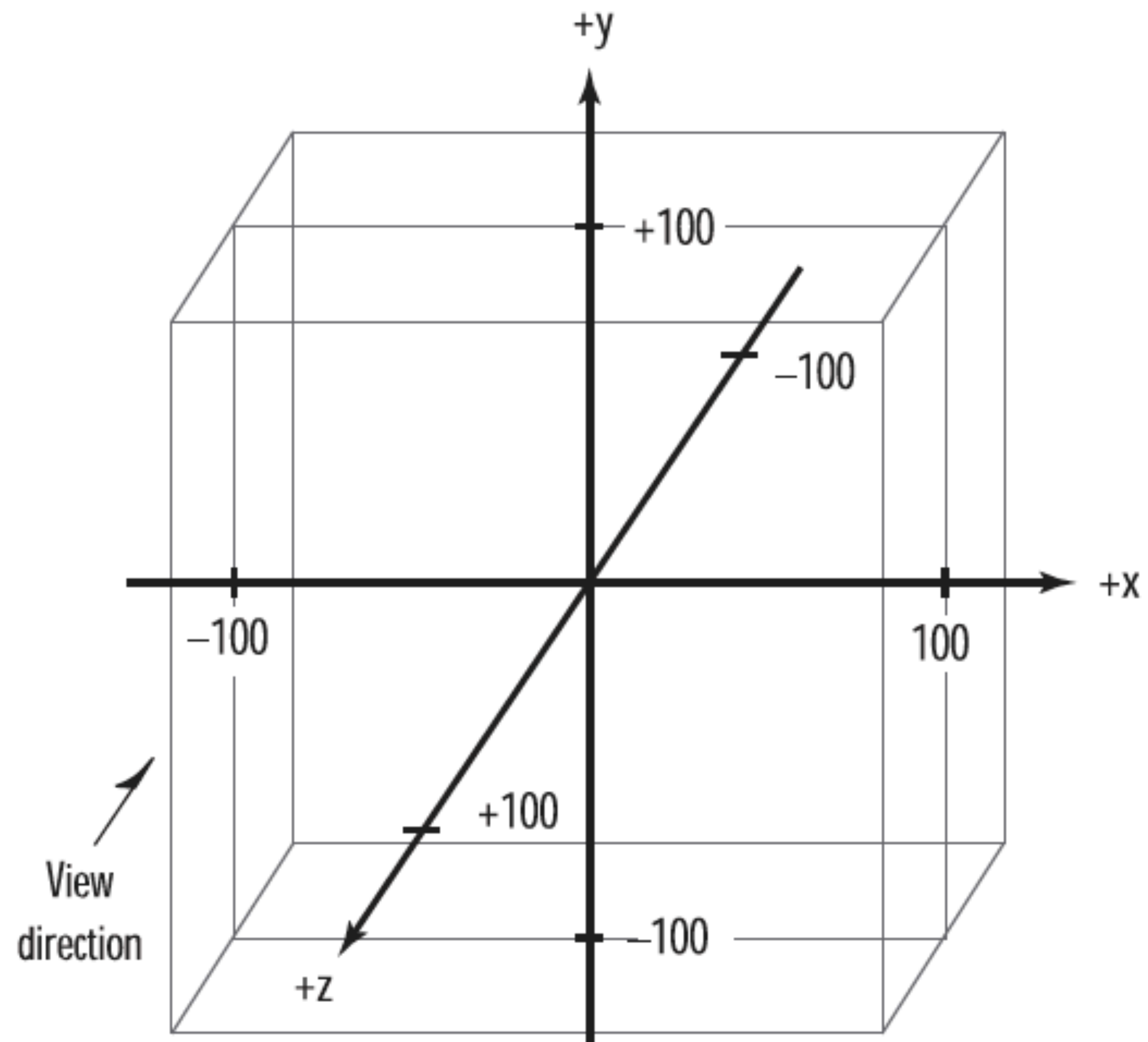
Pixels & Points

- Computer graphics at its simplest: Draw a point somewhere on the screen, and make it a specific color.
- Build on this simple concept, to produce lines, polygons, circles, and other shapes and graphics
- In OpenGL, we are not concerned with physical screen coordinates and pixels, but rather positional coordinates in a viewing volume.
- OpenGL worries about how to get your points, lines, etc projected from your established 3D space to the 2D image on a screen

Viewing Volume

- Consists of an area enclosed a Cartesian coordinate space that ranges from –100 to+100 on all three axes—x, y, and z.

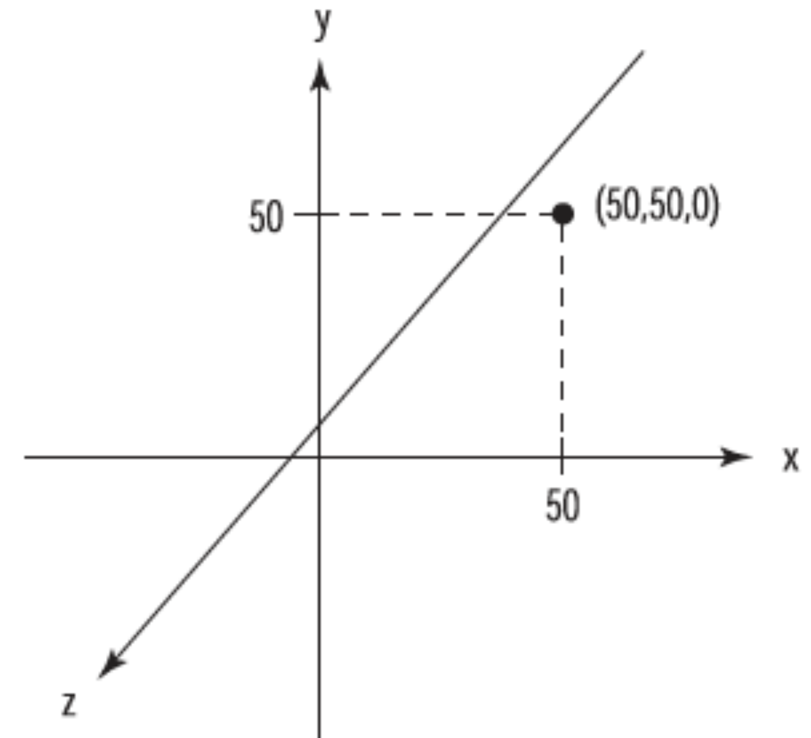
- We established this volume with a call to glOrtho.




```
void setupRC(void)
{
  glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
  glOrtho (-100.0f, 100.0f, -100.0f, 100.0f, -100.0f, 100.0f);
}
```

A 3D Point: The Vertex

- Function glVertex— one of the most used functions in all the OpenGL API.
- The “lowest common denominator” of all the OpenGL primitives: a single point in space.
- The glVertex function can take from one (a pointer) to four parameters of any numerical type,from bytes to doubles, subject to the naming conventions

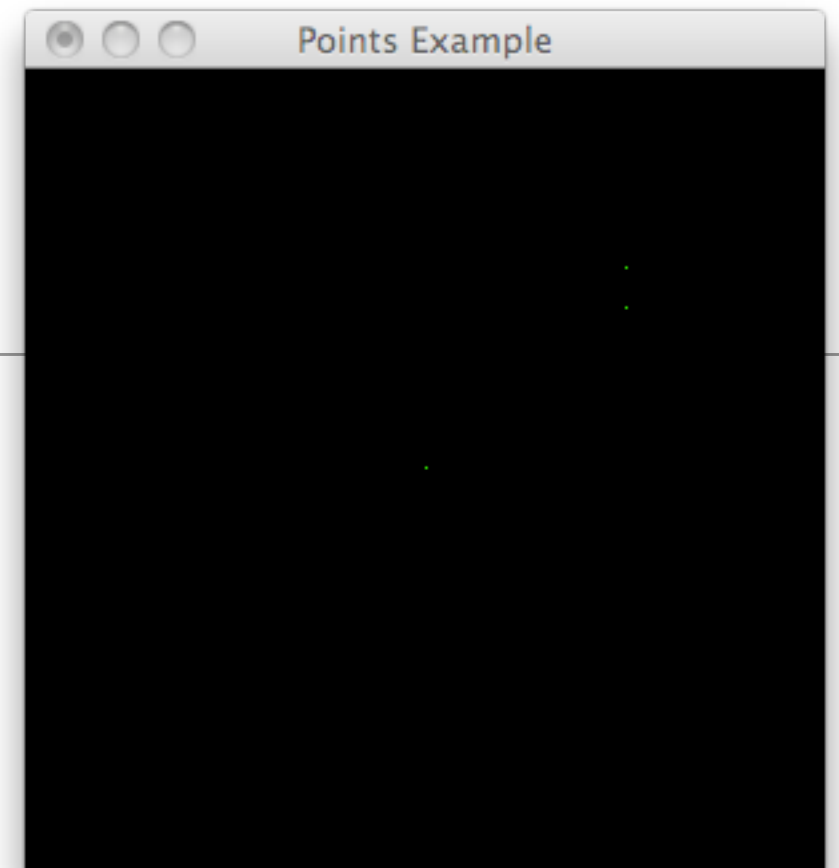



```
glVertex3f(50.0f, 50.0f, 0.0f);
```

Primitives

- Is this vertex a point that should just be plotted or is it the endpoint of a line or the corner of a cube?
- The geometric definition of a vertex is not just a point in space, but rather the point at which an intersection of two lines or curves occurs.
- This is the essence of primitives.
- A primitive is simply the interpretation of a set or list of vertices into some shape drawn on the screen.
- There are 10 primitives in OpenGL, from a simple point drawn in space to a closed polygon of any number of sides.

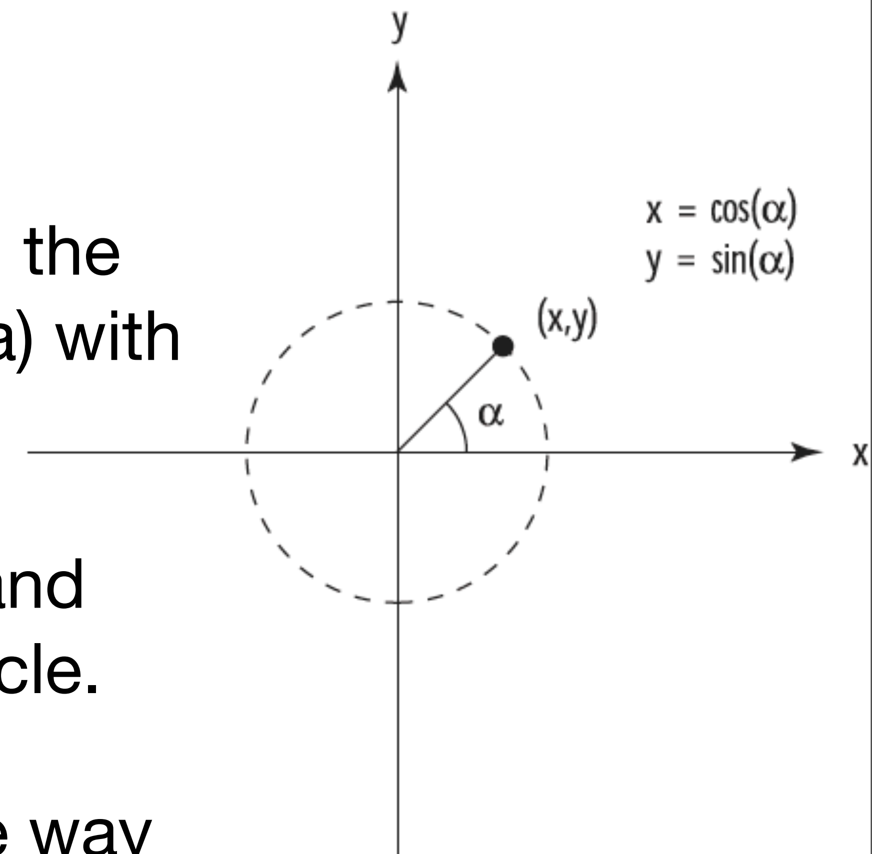
Drawing Points



- One way to draw primitives is to use the glBegin command to tell OpenGL to begin interpreting a list of vertices as a particular primitive.
- You then end the list of vertices for that primitive with the glEnd command.
- glBegin, GL_POINTS tells OpenGL that the succeeding vertices are to be interpreted and drawn as points.

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

  glBegin( GL_POINTS);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(50.0f, 40.0f, 0.0f);
    glVertex3f(50.0f, 50.0f, 50.0f);
  glEnd();

  glutSwapBuffers();
}
```

sin / cos



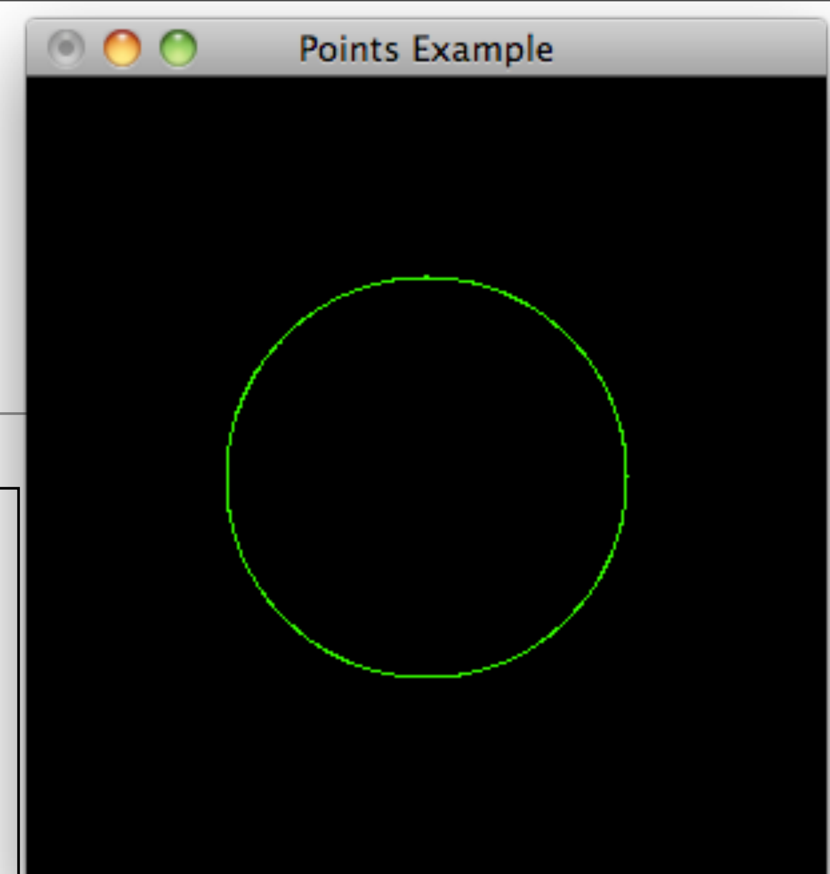
- A circle drawn in the xy plane and a line segment from the origin (0,0) to any point on the circle makes an angle (a) with the x-axis.

- For any given angle, the trigonometric functions sine and cosine return the x and y values of the point on the circle.

- By stepping a variable that represents the angle all the way around the origin, we can calculate all the points on the circle.

- The C runtime functions sin() and cos() accept angle values measured in radians instead of degrees. There are 2*PI radians in a circle.

Drawing a Circle




```
void renderScene(void)
{
  GLfloat x, y, angle;

  glClear( GL_COLOR_BUFFER_BIT);

  glBegin( GL_POINTS);
  for (angle = 0.0f; angle <= (2.0f * GL_PI); angle += 0.01f)
    {
      x = 50.0f * sin(angle);
      y = 50.0f * cos(angle);
      glVertex3f(x, y, 0.0f);
    }
  glEnd();

  glutSwapBuffers();
}
```

- Calculates the x and y coordinates for an angle that spins between 0°and 360°
- Expressed programmatically in radians rather than degrees

Setting the Point Size

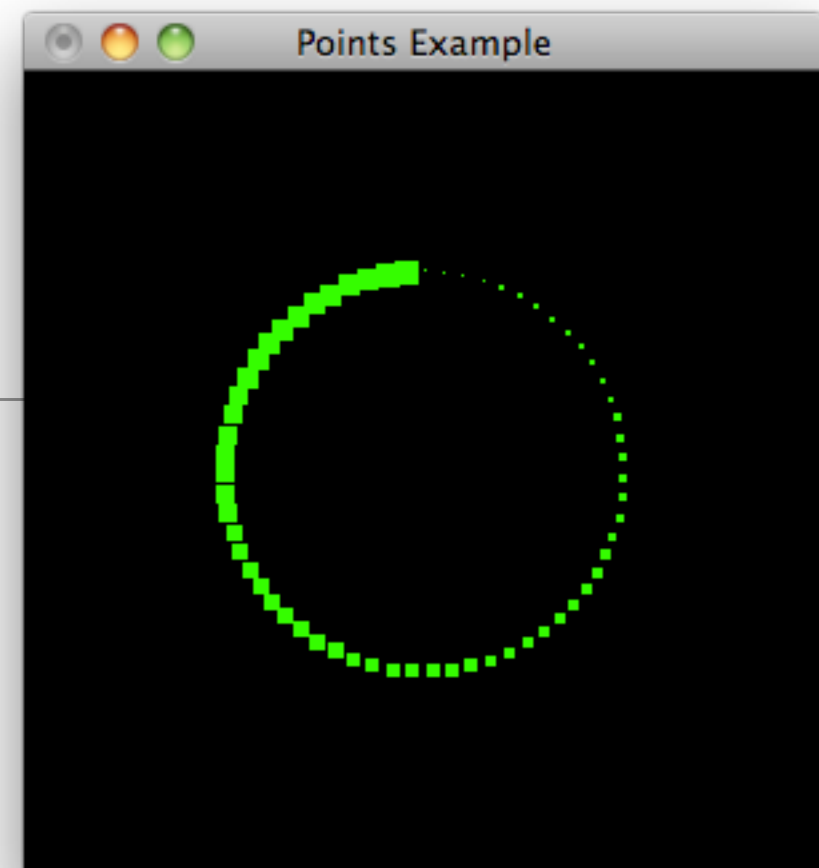
- When you draw a single point, the size of the point is one pixel by default.
- Change this size with the function glPointSize:

```
void glPointSize(GLfloat size);
```

- It specifies the approximate diameter in pixels of the point drawn.
- Not all point sizes are supported, however, and you should make sure the point size you specify is available

```
GLfloat sizes[2];
GLfloat step;
GLfloat curSize;


void retupRC()
{
  //...
  glGetFloatv(GL_POINT_SIZE_RANGE,sizes);
  glGetFloatv(GL_POI¯T_SIZE_GRANULARITY,&step);
  curSize = sizes[0];
  //...
}
```

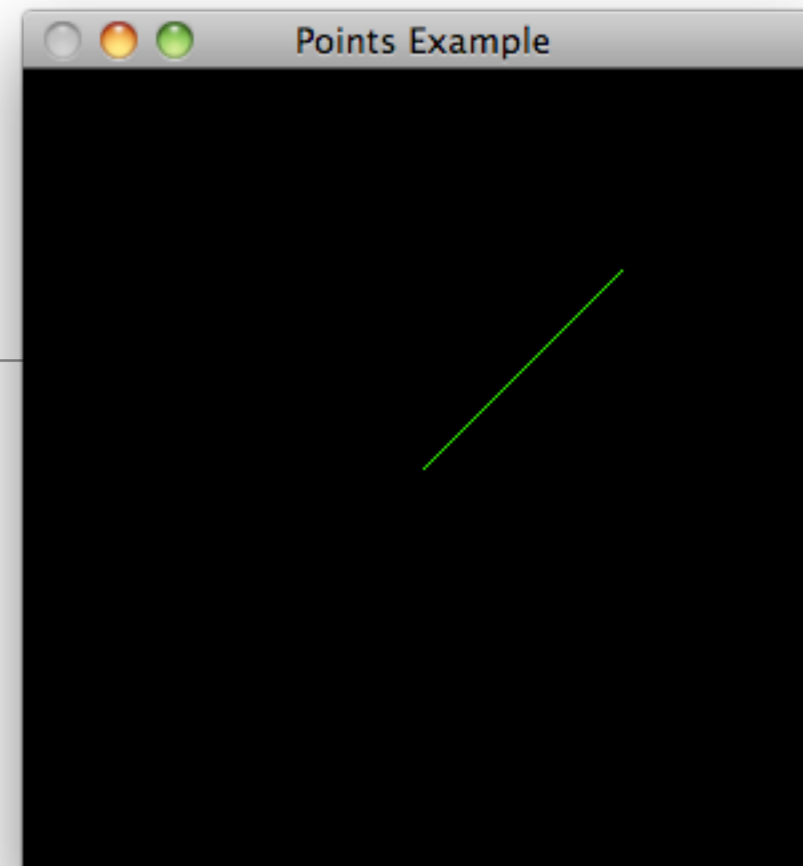
```
void renderScene(void)
{
  GLfloat x, y, angle;

  glClear(GL_COLOR_BUFFER_BIT);

  for (angle = 0.0f; angle <= (2.0f * GL_PI); angle += 0.1f)
    {
      x = 50.0f * sin(angle);
      y = 50.0f * cos(angle);
      glPointSize(curSize);
      glBegin( GL_POINTS);
        glVertex3f(x, y, 0.0f);
      glEnd();
      curSize+=step;
    }
  glutSwapBuffers();
}
```



- glPointSize must be called outside the glBegin/glEnd statements.
- Using a point size larger supported OpenGL uses the largest available point size but does not keep growing - the range are clamped to the range
- Larger point sizes are represented simply by larger cubes. This is the default behavior, but it typically is undesirable for many application.
- Need *Antialiasing*, a technique used to smooth out jagged edges and round out corners

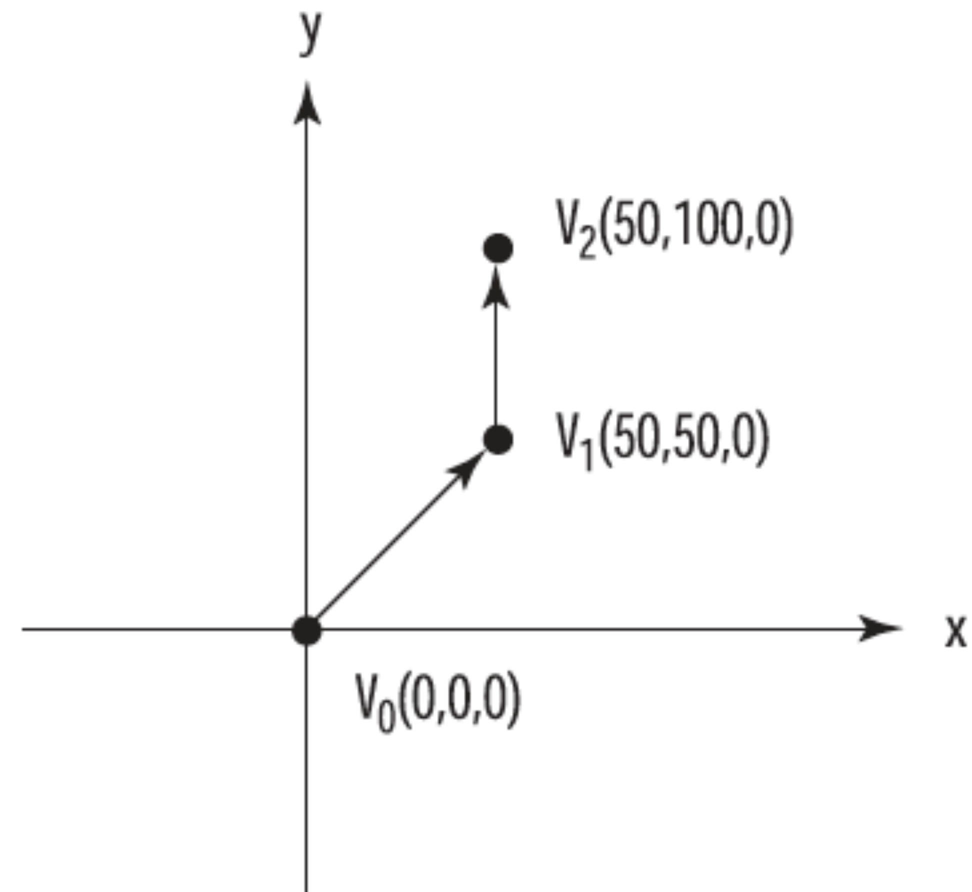
Drawing Lines

- GL_POINTS: for each vertex specified, it draws a point.
- GL_LINES: to specify two vertices and draw a line between them.
- two vertices specify a single primitive
- If you specify an odd number of vertices for GL_LINES, the last vertex is just ignored

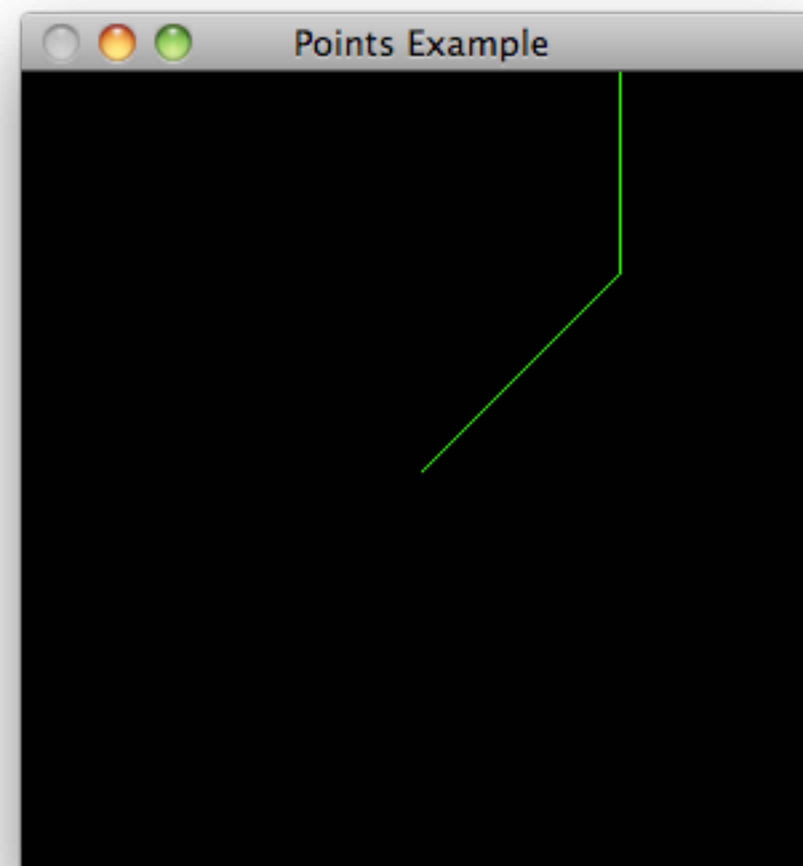



```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glBegin( GL_LINES);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(50.0f, 50.0f, 0.0f);
  glEnd();

  glutSwapBuffers();
}
```

Connected Lines

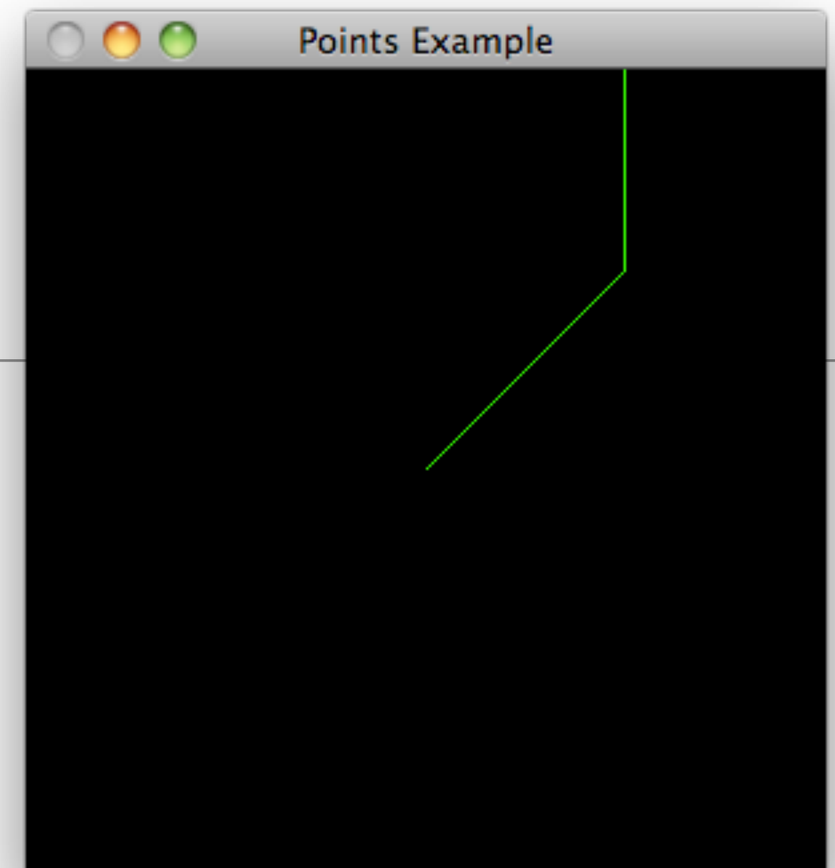



```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glBegin( GL_LINES);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(50.0f, 50.0f, 0.0f);
    glVertex3f(50.0f, 50.0f, 0.0f);
    glVertex3f(50.0f, 100.0f, 0.0f);
  glEnd();

  glutSwapBuffers();
}
```

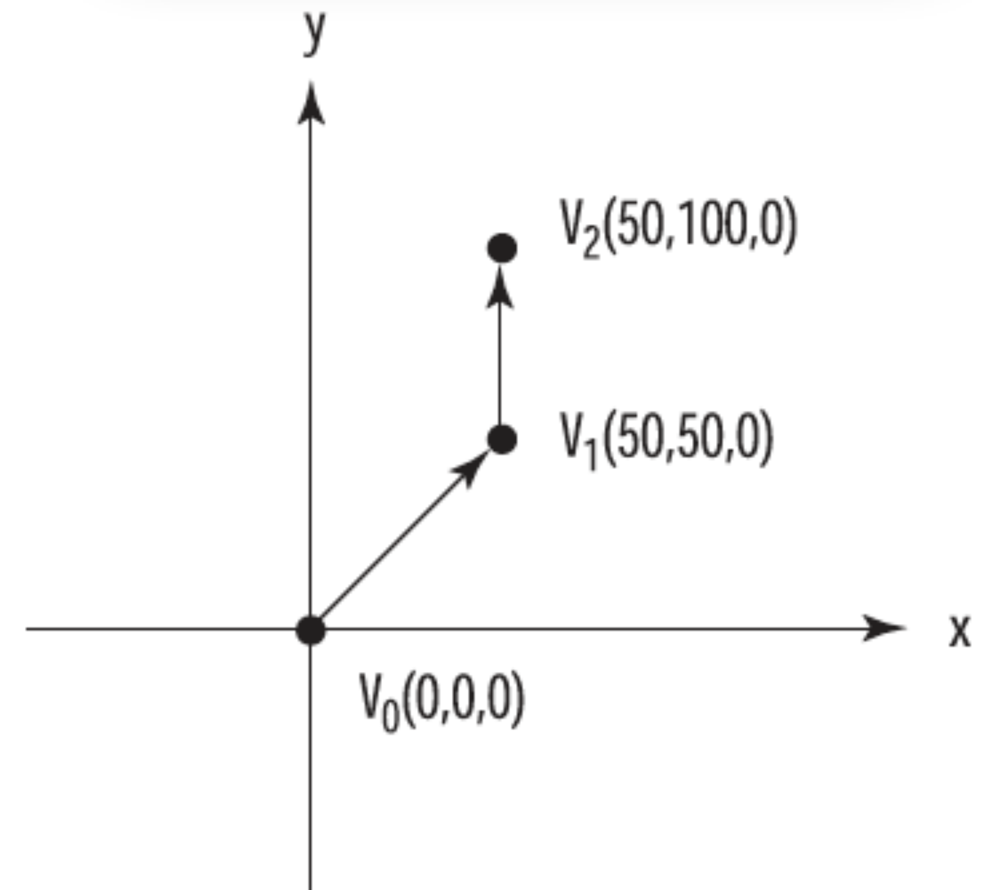


Line Strips

- GL_LINE_STRIP, a line is drawn from one vertex to the next in a continuous segment

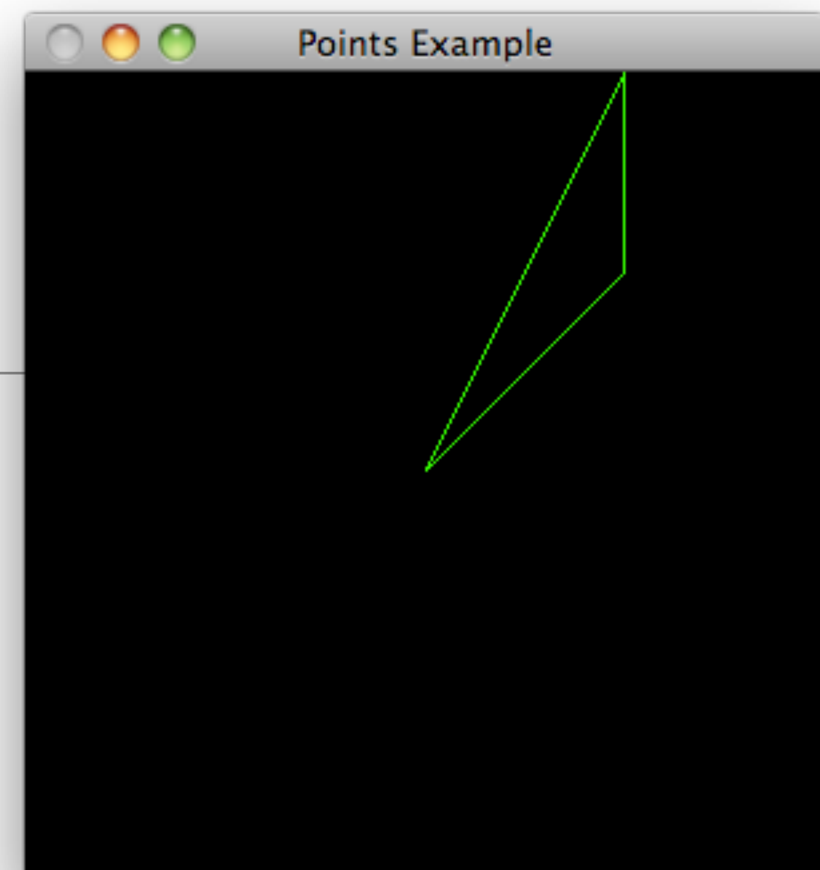



```
void renderScene(void)
{
  glClear( GL_COLOR_BUFFER_BIT);

  glBegin( GL_LINE_STRIP);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(50.0f, 50.0f, 0.0f);
    glVertex3f(50.0f, 100.0f, 0.0f);
  glEnd();

  glutSwapBuffers();
}
```

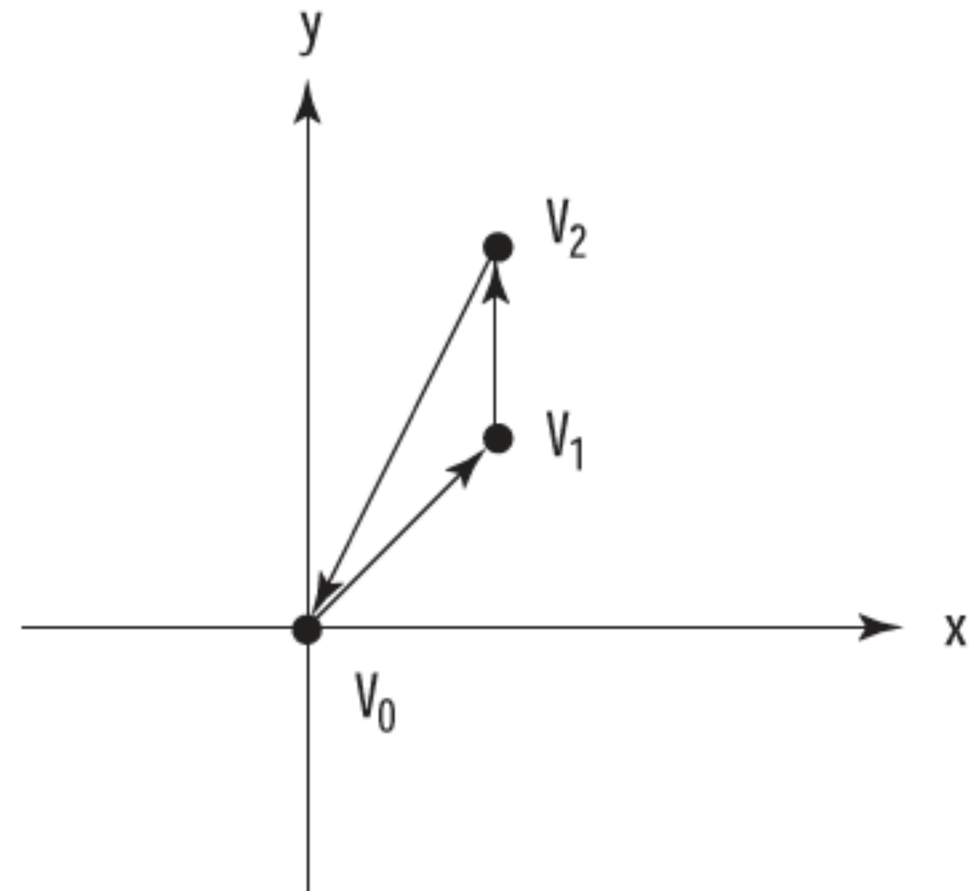


Line Loops

- LINE_LOOP: behaves just like GL_LINE_STRIP, but one final line is drawn between the last vertex specified and the first one specified




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

  glBegin( GL_LINE_LOOP);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(50.0f, 50.0f, 0.0f);
    glVertex3f(50.0f, 100.0f, 0.0f);
  glEnd();

  glutSwapBuffers();
}
```



Approximating Curves with Straight Lines

- Plot points along a circle-shaped path.
- Can push the points closer and closer together (by setting smaller values for the angle increment) to create a smooth curve instead of the broken points
- Can be slow for larger and more complex curves with thousands of points.

```
void renderScene(void)
{
  GLfloat x, y, angle;

  glClear( GL_COLOR_BUFFER_BIT);

  glBegin( GL_POINTS);
  for (angle = 0.0f; angle <= (2.0f * GL_PI); angle += 0.01f)
    {
      x = 50.0f * sin(angle);
      y = 50.0f * cos(angle);
      glVertex3f(x, y, 0.0f);
    }
  glEnd();

  glutSwapBuffers();
}
```

Connect the dots

- Approximating a curve using GL_LINE_STRIP or GL_LINE_LOOP to connect-the-dots.
- As the dots move closer together, a smoother curve materializes without you having to specify all the points.

```
void renderScene(void)
{
  GLfloat x, y, angle;

  glClear( GL_COLOR_BUFFER_BIT);

  glBegin( GL_LINE_LOOP);
  for (angle = 0.0f; angle <= (2.0f * GL_PI); angle += 0.1f)
    {
      x = 50.0f * sin(angle);
      y = 50.0f * cos(angle);
      glVertex3f(x, y, 0.0f);
    }
  glEnd();

  glutSwapBuffers();
}
```

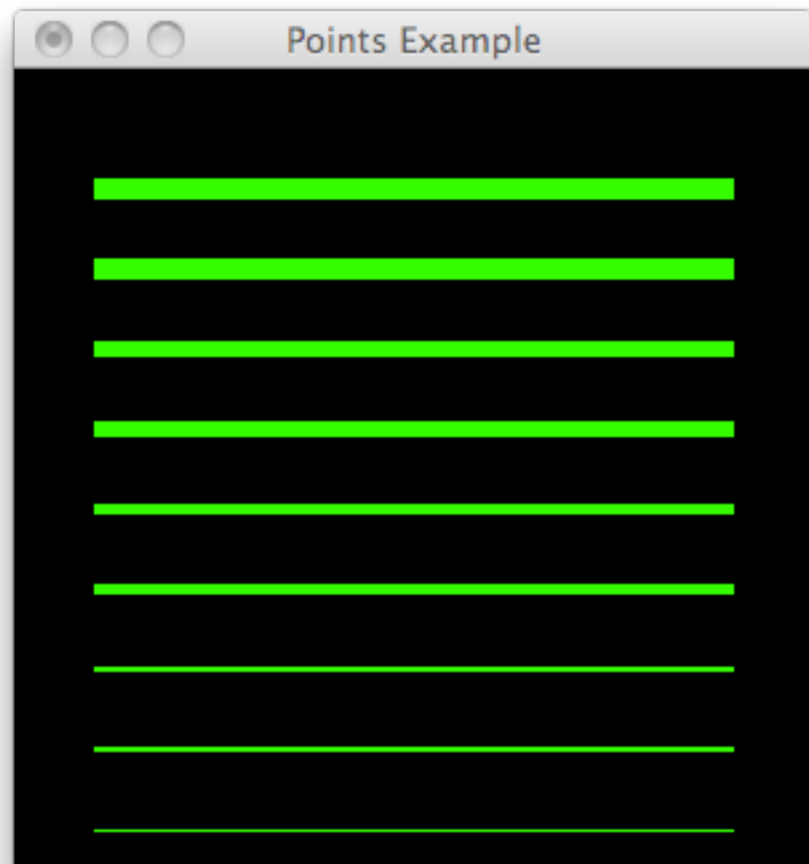
Setting the Line Width

- Specify various line widths when drawing lines by using the glLineWidth function.
- The glLineWidthfunction takes a single parameter that specifies the approximate width, in pixels, of the line drawn.
- Just as with point sizes, not all line widths are supported, and you should make sure that the line width you want to specify is available.

```
void glLineWidth (GLfloat width);
```

```
GLfloat fSizes[2];
GLfloat fCurrSize;

void retupRC()
{
  //...

  glGetFloatv(GL_LINE_WIDTH_RANGE,fSizes);
  fCurrSize = fSizes[0];
}
```

Lines Example

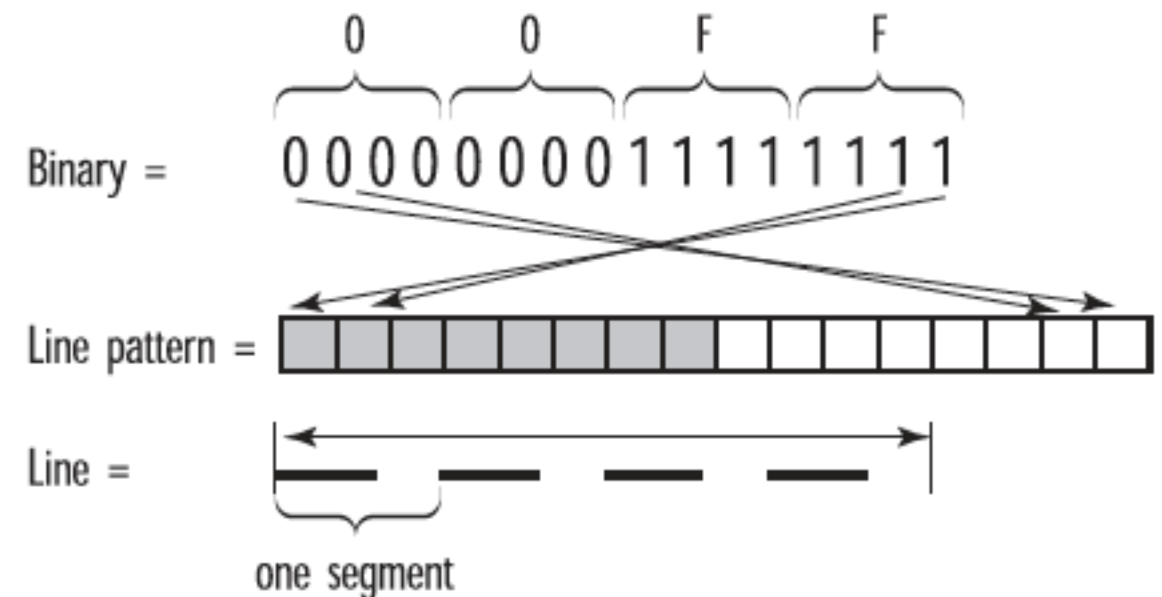



```
void renderScene(void)
{
 GLfloat y;

 glClear(GL_COLOR_BUFFER_BIT);

 for(y = -90.0f; y < 90.0f; y += 20.0f)
  {
   glLineWidth(fCurrSize);

   glBegin(GL_LINES);
      glVertex2f(-80.0f, y);
      glVertex2f(80.0f, y);
   glEnd();

   fCurrSize += 1.0f;
  }

 glutSwapBuffers();
}
```

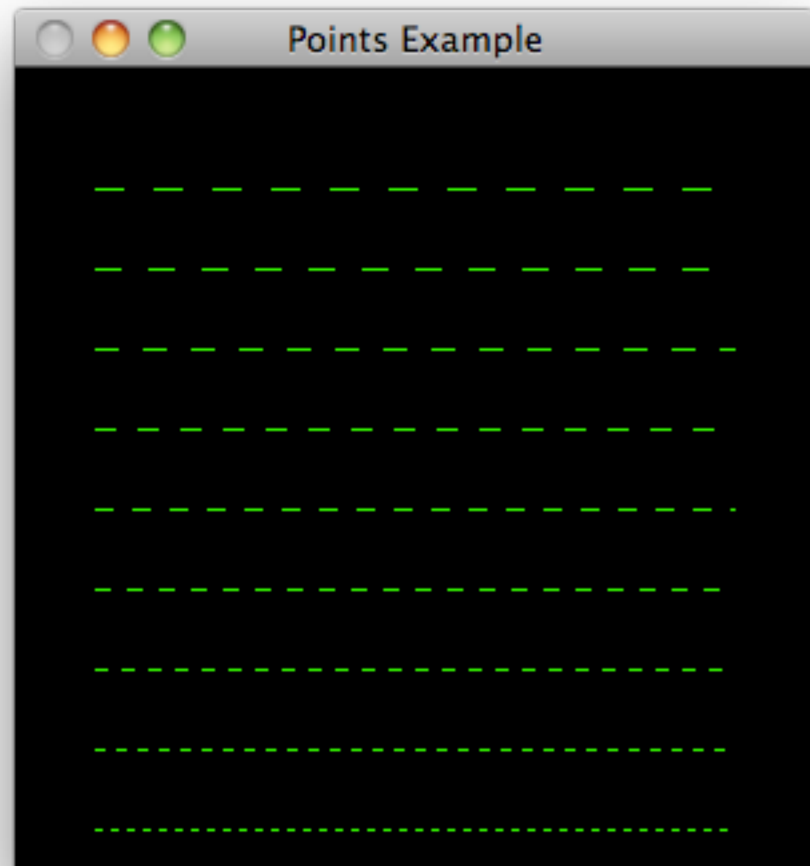
Line Stippling

```
void glLineStipple (GLint factor, GLushort pattern);
```

- In addition to changing line widths, you can create lines with a dotted or dashed pattern, called stippling
- The pattern parameter is a 16-bit value that specifies a pattern to use when drawing the lines.
- Each bit represents a section of the line segment that is either on or off.
- By default,each bit corresponds to a single pixel, but the factor parameter serves as a multiplier to increase the width of the pattern.



Stipple Example




```
void retupRC()
{
  //..

  glEnable(GL_LINE_STIPPLE);
}
```

```
void renderScene(void)
{
 GLfloat y;
 GLint factor = 3;
 GLushort pattern = 0x5555;

 glClear(GL_COLOR_BUFFER_BIT);

 for(y = -90.0f; y < 90.0f; y += 20.0f)
  {
   glLineStipple(factor,pattern);

    glBegin(GL_LINES);
      glVertex2f(-80.0f, y);
      glVertex2f(80.0f, y);
    glEnd();

   factor++;
  }
 glutSwapBuffers();
}
```