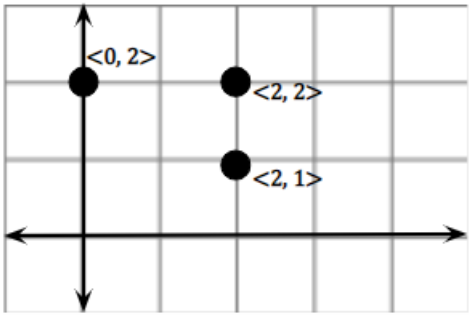
1 Three Paradigms for Rendering

We first begin by discussing how to render points, lines, and triangles.

1.1 Drawing Points

Consider the following graph:



In this class, we represent points in the form $\langle x, y \rangle$, or $\begin{bmatrix} x \\ y \end{bmatrix}$. In C++, we can represent these points like so:

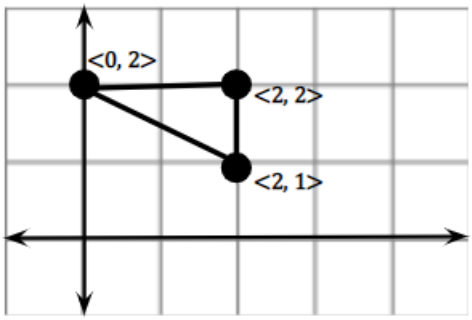
```
float verts[][2] = {
    {2, 1},
    {2, 2},
    {0, 2}
};
```

After loading the array into the GPU, which we'll discuss later, we can use a command like:

```
glDrawArrays(GL_POINTS, 0, 3);
```

1.2 Drawing Lines

Consider the following graph:



In JavaScript, we can make use of the Canvas API to draw this like so:

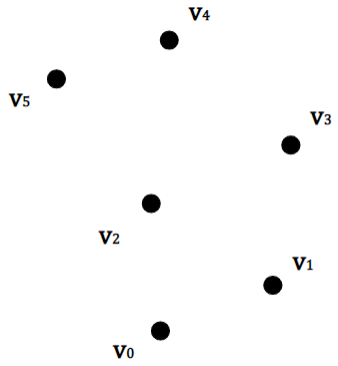
```
moveTo(2, 1);
lineTo(2, 2);
lineTo(0, 2);
lineTo(2, 2);
stroke();
```

In OpenGL, we would do something like:

```
glDrawArrays(GL_LINE_LOOP, 0, 3);
```

The `GL_LINE_LOOP` means that we're making a closed loop of edges, using 3 vertices in total.

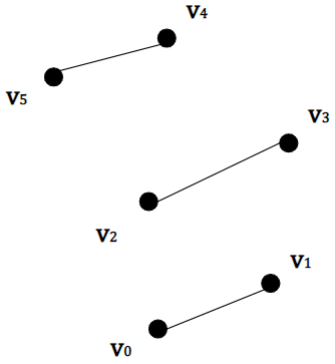
There are several other modes that we can use. To see how these differ, we'll use the following set of vertices as an example:



- `GL_LINES`: If we have the following code segment

  ```
  glDrawArrays(GL_LINES, 0, 6);
  ```

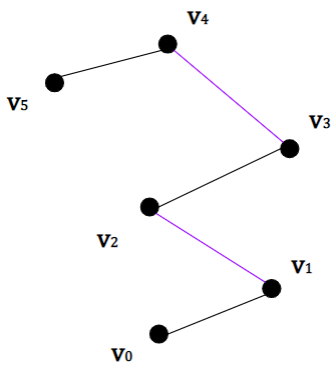
  Then, we would get the following drawing:



- `GL_LINE_STRIP`: Now, if we were to include the following code segment in addition to the one above

  ```
  glDrawArrays(GL_LINES_STRIP, 0, 6);
  ```

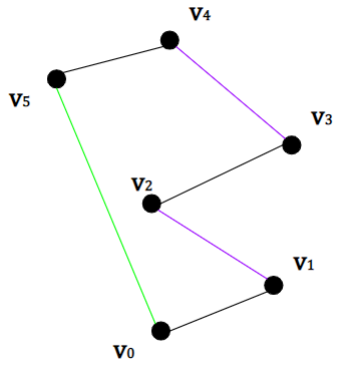
  Then, we would get the following drawing:



- `GL_LINE_LOOP`: Finally, including

  ```
  glDrawArrays(GL_LINES_LOOP, 0, 6);
  ```

  would yield the following drawing:



So, to sumamrize, if we have the vertices $\{v_1, v_2, \ldots, v_n\}$, then:

- `GL_LINES` will draw a line for each pair of vertices; that is, a line will be drawn between v_1 and v_2, v_3 and v_4, and so on.
- `GL_LINE_STRIP` will draw a line for each consecutive pair of vertices, up to and including v_{n-1}; that is, a line will be drawn between v_1 and v_2, v_2 and v_3, v_3 and v_4, and so on. The last line drawn will be from v_{n-1} to v_n.
- `GL_LINE_LOOP` will draw a line for each consecutive pair of vertices, including from the end vertex to the start vertex. So, effectively, this is just `GL_LINE_STRIP` but with a line from v_n to v_1.

1.3 Drawing Triangles

Like with drawing lines, there are three modes for drawing triangles.

- `GL_TRIANGLES`
- `GL_TRIANGLE_FAN`
- `GL_TRIANGLE_STRIP`

Using the same set of 6 points above, we show the following examples.

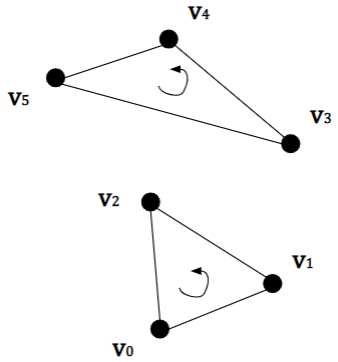
1.3.1 `GL_TRIANGLES` Mode

This mode groups the vertices into groups of three, and then draws a triangle between each group. For example, if you have vertices $\{v_0, \ldots, v_5\}$, then this mode would take vertices $\{v_0, \ldots, v_2\}$ and $\{v_3, \ldots, v_5\}$ and draw a triangle (from $v_0 \to v_1$ and then from $v_1 \to v_2$ and finally $v_2 \to v_0$, while filling it in with a color).

When we use the `GL_TRIANGLES` mode, like so:

```
glDrawArrays(GL_TRIANGLES, 0, 6);
```

Then we get something like:

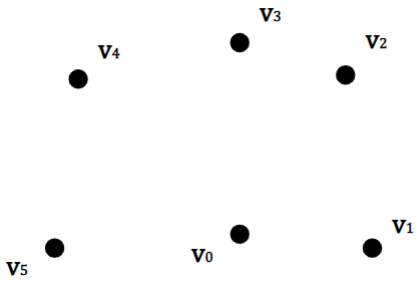


Note that the arrows here indicate that we're looking at the "front" faces of the triangle. The "back" face is the other side.

1.3.2 `GL_TRIANGLE_FAN` Mode

For an array of vertices $\{v_0, \ldots, v_n\}$, v_0 is the common vertex. Then, the rest of the vertices $v_1, \ldots, v_n$ are defining a triangle which shares the initial vertex.

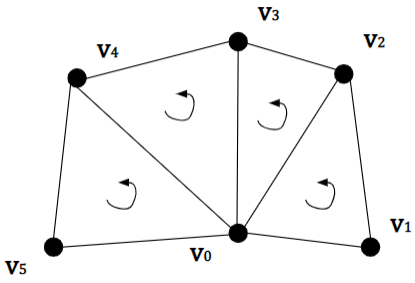
Suppose you're given the following set of vertices like so:



When using this mode, like so

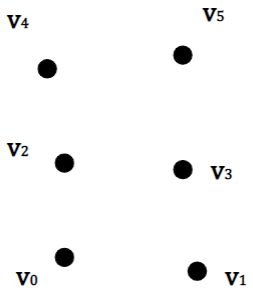
```
glDrawArrays(GL_TRIANGLE_FAN, 0, 6);
```

Then we get something like:



1.3.3 `GL_TRIANGLE_STRIP` Mode

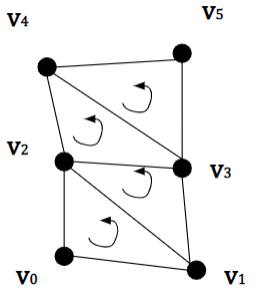
Suppose you're given the following set of vertices like so:



When using this mode, like so

```
glDrawArrays(GL_TRIANGLE_STRIP, 0, 6);
```

Then we get something like:



The way to think about this is that you have your "base" triangle with vertices $\{v_0, v_1, v_2\}$. Then, v_3 is facing the edge formed between v_1 and v_2. Likewise, the vertex v_4 is facing the edge formed between v_2 and v_3 of the triangle with vertices $\{v_1, v_2, v_3\}$.