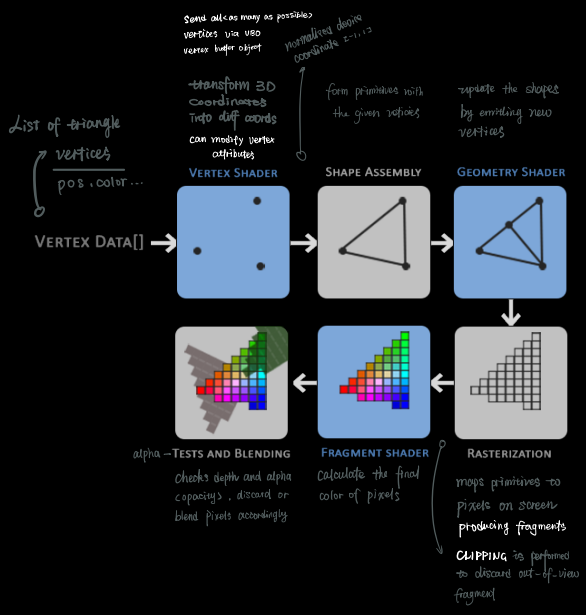

Our vertex buffer data is formatted as follows:

- The position data is stored as 32-bit (4 byte) floating point values.
- Each position is composed of 3 of those values.
- There is no space (or other values) between each set of 3 values. The values are tightly packed in the array.
- The first value in the data is at the beginning of the buffer.

With this knowledge we can tell OpenGL how it should interpret the vertex data (per vertex attribute) using `glVertexAttribPointer`:

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
```

The function `glVertexAttribPointer` has quite a few parameters so let's carefully walk through them:

- The first parameter specifies which vertex attribute we want to configure. Remember that we specified the location of the `position` vertex attribute in the vertex shader with `layout (location = 0)`. This sets the location of the vertex attribute to 0 and since we want to pass data to this vertex attribute, we pass in 0.
- The next argument specifies the size of the vertex attribute. The vertex attribute is a `vec3` so it is composed of 3 values.
- The third argument specifies the type of the data which is `GL_FLOAT` (a `vec*` in GLSL consists of floating point values).
- The next argument specifies if we want the data to be normalized. If we're inputting integer data types (int, byte) and we've set this to `GL_TRUE`, the integer data is normalized to 0 (or -1 for signed data) and 1 when converted to float. This is not relevant for us so we'll leave this at `GL_FALSE`.
- The fifth argument is known as the stride and tells us the space between consecutive vertex attributes. Since the next set of position data is located exactly 3 times the size of a `float` away we specify that value as the stride. Note that since we know that the array is tightly packed (there is no space between the next vertex attribute value) we could've also specified the stride as 0 to let OpenGL determine the stride (this only works when values are tightly packed). Whenever we have more vertex attributes we have to carefully define the spacing between each vertex attribute but we'll get to see more examples of that later on.
- The last parameter is of type `void*` and thus requires that weird cast. This is the offset of where the position data begins in the buffer. Since the position data is at the start of the data array this value is just 0. We will explore this parameter in more detail later on