Let us follow a single vertex through all the steps of the rendering pipeline, from its original Model coordinates to its final index in the FrameBuffer's pixel-array.

1. **Model-to-Camera Transformation** (Model Coordinates to Camera Coordinates)

   Let (x_m, y_m, z_m) be a `Vertex` in a `Model`'s coordinate system and let (x_t, y_t, z_t) be the translation `Vector` that accompanies the `Model` in a `Position`. Then the vertex's camera coordinates are given by

   $$x_c = x_m + x_t, \quad y_c = y_m + y_t, \quad z_c = z_m + z_t$$

2. **Projection Transformation** (Camera Coordinates to Image-plane Coordinates)

   Let (x_c, y_c, z_c) be a vertex in camera coordinates and let (x_{ip}, y_{ip}, z_{ip}) be its perspective projection onto the camera's image-plane. Then

   $$x_{ip} = -x_c/z_c,$$
   $$y_{ip} = -y_c/z_c,$$
   $$z_{ip} = -1.$$

   Vertices in camera coordinates that are within the camera's view volume will project to vertices in the image-plane's *view rectangle* with coordinates that satisfy

   $$-1 \le x_{ip} \le 1 \quad \text{and} \quad -1 \le y_{ip} \le 1.$$

3. **Image-plane to Pixel-plane Transformation**

   Let $(x_{ip}, y_{ip}, -1)$ be a vertex in the camera's image-plane and let (x_{pp}, y_{pp}) be its transformation to the renderer's pixel-plane. Then

   $$x_{pp} = 0.5 + (w_{vp}/2.001)(x_{ip} + 1),$$
   $$y_{pp} = 0.5 + (h_{vp}/2.001)(y_{ip} + 1).$$

   where w_{vp} and h_{vp} are the width and height of the `FrameBuffer Viewport` that we are rendering into. A vertex $(x_{ip}, y_{ip}, -1)$ from the image-plane's view rectangle will transform to a two-dimensional vertex (x_{pp}, y_{pp}) in the renderer's *logical viewport* with coordinates that satisfy

   $$0.5 \le x_{pp} < w_{vp} + 0.5 \qquad \text{and} \qquad 0.5 \le y_{pp} < h_{vp} + 0.5.$$

   Points in the pixel-plane with integer coordinates are called *logical pixels*.

4. **Pixel-plane to Viewport Transformation**

   Let $(x_{pp},\, y_{pp})$ be a vertex in the renderer's logical viewport (in the pixel-plane). Then

   $$(\texttt{Math.round}(x_{pp}),\, \texttt{Math.round}(y_{pp}))$$

   is the logical pixel nearest to $(x_{pp},\, y_{pp})$. Let $(x_{vp},\, y_{vp})$ be its equivalent (physical) pixel in the `FrameBuffer`'s `Viewport`. Then

   $$x_{vp} = (\texttt{int})\texttt{Math.round}(x_{pp}) - 1,$$

   $$y_{vp} = h_{vp} - (\texttt{int})\texttt{Math.round}(y_{pp}).$$

   Pixels $(x_{vp},\, y_{vp})$ in a `Viewport` have integer coordinates that should satisfy

   $$0 \le x_{vp} \le w_{vp} - 1 \qquad \text{and} \qquad 0 \le y_{vp} \le h_{vp} - 1$$

   with the pixel (0, 0) being the upper left-hand corner of the `Viewport`. If a pixel does not satisfy these bounds, then that pixel should be *clipped* (not entered into the `Viewport`).

5. **Viewport to FrameBuffer**

   Suppose that a `Viewport`'s upper left-hand corner in the `FrameBuffer` is at $(x_{ul},\, y_{ul})$. Let $(x_{vp},\, y_{vp})$ be a pixel using `Viewport` coordinates. Then that pixel's coordinates in the `FrameBuffer` are given by

   $$x = x_{ul} + x_{vp}, \quad y = y_{ul} + y_{vp}.$$

   Note: The `FrameBuffer` will use this formula even when the pixel's `Viewport` coordinates are not within the `Viewport`'s width and height. This will lead to either the pixel appearing outside of the `Viewport` or the pixel appearing misplaced in the `Viewport` or to an `ArrayIndexOutOfBoundsException`.

6. **FrameBuffer to pixel-array**

   Suppose that a `FrameBuffer` has width w and height h. The `FrameBuffer`'s pixel data is stored in a one-dimensional, row-major, array `int[w * h]` that we will call the *pixel-array*. Let $(x,\, y)$ be a pixel using `FrameBuffer` coordinates. Its index in the pixel-array is given by

   $$\texttt{index} = \texttt{y} * \texttt{w} + \texttt{x}.$$

   Note: The `FrameBuffer` will use this formula even when the pixel's `FrameBuffer` coordinates are not within the `FrameBuffer`'s width and height. This will lead to either the pixel appearing misplaced in the `FrameBuffer` or to an `ArrayIndexOutOfBoundsException`.