

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_buffer`.

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

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

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

$$x_{pp} = 0.5 + (w_{vp}/2.0)(x_{ip} + 1),$$

$$y_{pp} = 0.5 + (h_{vp}/2.0)(y_{ip} + 1).$$

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 inside the image-plane's view rectangle will transform to a two-dimensional vertex (x_{pp}, y_{pp}) inside the renderer's *logical viewport* with coordinates that satisfy

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

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

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

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

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

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

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

$$0 \leq x_{vp} \leq w_{vp} - 1 \quad \text{and} \quad 0 \leq y_{vp} \leq h_{vp} - 1$$

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_buffer

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 call the **pixel_buffer**. Let (x, y) be a pixel using **Framebuffer** coordinates. Its index in the **pixel_buffer** is given by

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

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**.