Recall the Z-buffer algorithm:

for each polygon
    for each pixel indexed by i, j
        P ← point on polygon at this pixel
        if (depth(P) < depth[i][j])
            depth[i][j] ← depth(P)
            color[i][j] ← color(P)
        end if
    end for
end if

Assume right-handed eye space.
The point P projected onto the image plane (i.e. near plane) is
Q = {-(fPx)/Pz, -(fPy)/Pz}
The depth is
depth(P) = sqrt(Px^2, Py^2, Pz^2)

This is an expensive calculation for every fragment (what is a fragment?), so we replace it with an easier calculation, using “pseudodepth” (δ).

What about just using Pz, the distance from the plane perpendicular to the view vector through the eye?
    If we keep a consistent denominator among Qx, Qy, and Qδ (pseudodepth), we can take advantage of homogeneous coordinates to encode perspective projection as a 4x4 matrix.

The goals for pseudodepth:
The denominator should be - Pz
The numerator should be easy to compute, i.e. a linear function of Pz
The pseudodepth should be -1 for points on the near plane. Why?
The pseudodepth should be 1 for points on the far plane. Why?

So pseudodepth(-Pz) = (a Pz + b)/(- Pz) for some a, b
Given goals, 3, 4, solve for a, b:
    a = -(F + f) / (F – f)
    b = -2ff / (F – f)

Note the following:
Pseudodepth is a nonlinear function of Pz.
Points closer to the near plane have the highest pseudodepth resolution.
Points closer to the far plane have the lowest pseudodepth resolution.

Resulting perspective transformation matrix:

\[
Q = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \times \quad P = \begin{bmatrix} fP_x \\ fP_y \\ aP_z + b \\ -P_z \end{bmatrix}
\]

This still needs to take into account remapping \( P_x \) and \( P_y \) to the normalized view volume, but this is sufficient for the remaining discussion.

What is \( Q \) after perspective divide? Denote it \( Q = [Q_x, Q_y, Q_\delta, 1] \)

Using pseudodepth during scan conversion:

When scan converting a polygon, the linear nature of the pseudodepth numerator can be used to avoid calculating the perspective transformation of each fragment.

As a single example, consider the scan conversion code for a horizontal line \((P_0, P_1)\) in eye space and its perspective transformation \((Q_0, Q_1)\):

![Diagram of horizontal line and perspective transformation](image)

The eye space line is parameterized by \( s \), and the perspective transformation is parameterized by \( t \).

Here is scan conversion code that includes an incremental update of the pseudodepth \( \delta \):

\[
\delta \leftarrow Q_0 \delta \\
\text{delta} \delta \leftarrow (Q_1z - Q_0z) / (Q_1x - Q_0x) \\
\text{for } x \leftarrow Q_0x \text{ to } Q_1x \\
\quad \delta += \text{delta} \delta \\
\quad \text{if } (\delta < \text{depth}(x, y)) \\
\quad \quad \text{update depth and color buffers} \\
\quad \text{end if} \\
\text{end for}
\]

This speed-up extends straightforwardly for nonhorizontal lines, and also to polygons as with normal scan conversion.
Why does this work?

Assume \( a = 0, b = 1, f = 1 \)
Consider \( t \) in \([0, 1]\), \( \delta T = 1/(Q1x - Q0x) \)
and \( s \) in \([0, 1]\), along eye space line
and corresponding incremental steps along \( \delta \), as above.

We need to show \( Q\delta(t) \) is a linear function of \( t \). What is \( Q\delta(t) \)?
\[
Q\delta(t) = -1/Pz(s) = 1 / (P0z*(1-s) + P1z*s) = 1 / (P0z + s * (P1z – P0z))
\]

What is \( s \)? For every point \( P(s) \), there is a corresponding projection \( Q(t) \). In other words, for each \( s \) there is a corresponding \( t \). However, the correspondence is complicated—this derivation finds that correspondence.

Consider the mapping of \( Q(t) \) to \( P(s) \):
\[
Qx(t) = - Px(s) / Pz(s); \text{ solve for } s:
\]
\[
Qx(t)*P0z – Qx(t)*P0z*s + Qx(t)*P1z*s
= -P0x + P0x*s – P1x*s
\]
\[
Qx(t)*P0z + P0x = s* [Qx(t) * P0z – Qx(t) * P1z + P0x – P1x]
\]
\[
s = (Qx(t)*P0z + P0x) / (Qx(t)*(P0z – P1z) + P0x – P1x)
\]

Going back to the previous expression for \( Q\delta(t) \):
\[
Q\delta(t) = 1 / \{P0z + [Qx(t)*P0z + P0x] / (Qx(t)*(P0z – P1z) + P0x – P1x)] \}
\]
\[
= [Qx(t)*(P0z – P1z) + P0x – P1x] / [P0z*(Qx(t)*(P0z – P1z) + P0x – P1x) + (Qx(t)*P0z + P0x)*(P1z – P0z)]
\]
\[
= Qx(t) * (P0z – P1z)/(P0x*(P1z – P0z)) + (P0x-P1x) / (P0x*(P1z-P0z))
\]

\( Qx(t) \) is the only term that changes with respect to \( t \), and it is also linear with respect to \( t \), so \( Q\delta(t) \) is linear, hence the pseudodepth is linear in image space.

Why not increment \( Pz \), the eye space depth?

\( Pz \) is not a linear function of image space. From above, the image space point \( \{Qx, Qy\} \) is \( \{-fPx/Pz, -fPy/Pz\} \), a nonlinear relationship between both \( Qx \) and \( Qy \) (the scan-converting domain) and \( Pz \).
Why isn't Z linear in screen-space? Considered a foreshortened view of a building (or a checkerboard), which has windows along the side. The windows close by occupy many pixels, the ones far away are tiny. Stepping along pixels will correspond to small steps in Z for the nearby windows, and large steps in Z for the faraway windows.

For a triangle a great deal of computation can be saved:

- Incrementally update pDepth along two sides as scan-conversion progresses along scan lines. Scan convert each scan line between these two sides: