Homogeneous Coordinates

Homogeneous Coordinates: Adding a 4th Value to an XYZ Triple

We usually think of a 3D point as being represented by a triple: (x,y,z). Using homogeneous coordinates, we add a 4th number: (x,y,z,w).

A graphics system, by convention, performs transformations and clipping using (x,y,z,w) and then divides x, y, and z by w before it uses them.

\[
\begin{align*}
X &= \frac{x}{w} \\
Y &= \frac{y}{w} \\
Z &= \frac{z}{w}
\end{align*}
\]

Thus (1,2,3,1), (2,4,6,2), (-1,-2,-3,-1) all represent the same 3D point.

Homogeneous Coordinates: This Seems Awkward – Why Do It?

One reason is that it allows for perspective division within the matrix mechanism. The OpenGL call `glFrustum(left, right, bottom, top, near, far)` creates this matrix:

\[
\begin{pmatrix}
2 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -z & 1
\end{pmatrix}
\]

This gives \( w' = -z \), which is the necessary divisor for perspective.

Another Reason is to be able to Represent Points at Infinity

This is useful to be able specify a parallel light source by placing the light source location at infinity. The point (1,2,3,1) represents the 3D point (1,2,3) and the point (1,2,3,.5) represents the 3D point (2,4,6).

Points-at-infinity are used for parallel light sources and some shadow algorithms.

However, When Using Homogeneous Coordinates, You Sometimes Just Need to be able to get a Vector Between Two Points

To get a vector between two homogeneous points, we subtract them:

\[
\begin{align*}
(x_1, y_1, z_1, w_1) - (x_2, y_2, z_2, w_2) &= \left(\frac{x_1}{w_1}, \frac{y_1}{w_1}, \frac{z_1}{w_1}\right) - \left(\frac{x_2}{w_2}, \frac{y_2}{w_2}, \frac{z_2}{w_2}\right) \\
&= \left(\frac{w_2 x_1 - w_1 x_2}{w_1 w_2}, \frac{w_2 y_1 - w_1 y_2}{w_1 w_2}, \frac{w_2 z_1 - w_1 z_2}{w_1 w_2}\right)
\end{align*}
\]

Unfortunately, most of the time that we do this, we only want a unit vector in that direction, not the full vector. So, we can ignore the denominator, and just say:

\[
\hat{v} = \text{normalize}(w_2 x_1 - w_1 x_2, w_2 y_1 - w_1 y_2, w_2 z_1 - w_1 z_2);
\]

Which `normalize` function?

```
vec3 normalize(vec3 v)
{
    vec3 n = v;
    vec4 w = vec4(1.0, 1.0, 1.0, 1.0);
    n = vec3(n.x/w.x, n.y/w.y, n.z/w.z);
    return n;
}
```