Perspective Projection

A little bit ago I was creating a 3D renderer from scratch for CS class, and I thought it would be a fun idea to try deriving the rendering and projection techniques myself mathematically. I don't quite know if this is how it's usually done in general (perhaps I could have overcomplicated some things?), but this is what I came up with.

We shall first describe the following variables that we have to work with:

- $\mathbf{c}$, the vector containing the camera position,
- $\hat{\mathbf{n}}$, the unit vector relative to the camera position that contains the direction of the camera,
- $\mathbf{q}$, the center of the projection plane,
- dist, the distance from the camera to the projection plane,
- width, the width of the projection plane,
- height, the height of the projection plane,

The projection plane is the plane with center $\mathbf{q} = \mathbf{c} + \mathsf{dist} \cdot \hat{\mathbf{n}}$ and is oriented to be orthogonal to $\hat{\mathbf{n}}$.

The desired outcome of the perspective projection algorithm is, given a point $\mathbf{p}$, to determine where $\mathbf{p}$ lies when projected on the plane, whether or not it should be rendered (i.e. does it lie in the width and height bounds and it is in front of the camera?), and where on the screen it should be rendered.

The first order of business is to project the point onto the plane. Note that the perspective projection of $\mathbf{p}$ lies on the ray $\mathbf{r}(t) = t\mathbf{c} + (1 - t)\mathbf{p} = \mathbf{c} + t(\mathbf{p} - \mathbf{c})$, so we must determine where this ray intersects with the plane.

Observe that the projection plane is the set of vectors $\mathbf{x}$ such that $\hat{\mathbf{n}} \cdot (\mathbf{x} - \mathbf{q}) = 0$. With this, we may substitute $\mathbf{r}(t)$ in for $\mathbf{x}$ to determine the intersection:

$$\hat{\mathbf{n}} \cdot (\mathbf{r}(t) - \mathbf{q}) = \hat{\mathbf{n}} \cdot (\mathbf{c} - \mathbf{q} + t(\mathbf{p} - \mathbf{c})) = 0.$$

From this, we obtain

$$t_p = \frac{\hat{\mathbf{n}} \cdot (\mathbf{q} - \mathbf{c})}{\hat{\mathbf{n}} \cdot (\mathbf{p} - \mathbf{c})} = \frac{\mathsf{dist}}{\hat{\mathbf{n}} \cdot (\mathbf{p} - \mathbf{c})}.$$

With this construction, values of t_p that are negative correspond to points behind the camera, so these can be thrown out.

Now that we have the projected point onto the plane $\mathbf{r}(t_p)$, we must figure out where on the plane this value falls to render it onto the screen. To do so, we shall construct an orthogonal basis for the plane, with the origin at the center of the plane, and then solve for the corresponding position of the points mapped in terms of the plane basis.

Through 90° rotations of the camera direction vector about respective axes (converting to spherical coordinates and then simply adding angles makes this a whole lot easier), we can obtain vectors $\mathbf{r}$ and $\mathbf{d}$, which represent the right and down orthonormal basis vectors of the projection plane. Our goal then is to solve the following matrix equation:

$$A\mathbf{v} = \mathbf{p} - \mathbf{q},$$

where $A = \begin{bmatrix} \mathbf{r} & \mathbf{d} \end{bmatrix}$. Although A is not square, we may multiply by its left psuedoinverse on both sides to see that

$$\mathbf{v} = \begin{bmatrix} x \\ y \end{bmatrix} = (A^T A)^{-1} A^T (\mathbf{p} - \mathbf{q}).$$

From this, we can map the coordinates onto our rendering canvas and cull any points that potentially lie out of our view.