15-462: Computer Graphics I

Feb 2, 2012 – Perspective Projection

As I did not use slides for class today, I have prepared a list of points that I would like you to take away from today's lecture. I welcome feedback on whether or not this is helpful.

After today's class, you should be able to:

- Derive a "flat" perspective projection matrix based on reasoning about similar triangles. If the camera is pointing along the z-axis and we are projecting onto the near clipping plane at distance n, this matrix is:

$$\begin{matrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$$

- Suppose the near clipping plane is at unit distance (i.e., n=1). What is the projection of a vertex at coordinates (10, 10, 10)? At coordinates (5, 5, 5)? At coordinates (10, 5, 5)? On the near clipping plane, with coordinates (3, 5, 1)? At a generic location ***v***?
- What is perspective division and when is it applied?
- What is it that makes this projection flat? All of your points should have projected to the plane z=1.

- The problem with this matrix is that it does not preserve any depth information. Here is a perspective projection matrix that at least preserves depth *ordering*:

$$\begin{matrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{matrix}$$

- Using this new matrix, and assuming the far clipping plane is at location f=10, what are the projections of the points given above? What can you say about how the z coordinate of the projected points varies in relation to their original depths?
- The main thing to note is that depth ordering is preserved, the original depths are correct for the near and far clipping planes, and that depths in between those two clipping planes vary in a nonlinear fashion.
- Bonus: prove that we cannot design any 4x4 matrix that performs perspective projection and will preserve the original depths (z-coordinates) for all points.

- OpenGL buffers: OpenGL has a number of buffers that store information of interest in an array with width and height equal to the number of displayed pixels in the x and y directions respectively.
- The color buffer has R (red), G (green), B (blue), and alpha (transparency) components.

- There are front and back color buffers so that we may construct an image in the back buffer while the image in the front buffer is displayed. This construction allows us to avoid showing incomplete images or jumpy animations.
- A depth buffer is used to assist in rendering objects so that opaque objects occlude other objects behind them. As a triangle is rasterized for display, the depth buffer is queried to determine whether any other object has already been rendered that is closer to the camera. This query is performed pixel by pixel. If there is no closer object, the current triangle's color is placed in the current color buffer and its depth is placed in the depth buffer. In OpenGL, the depth buffer must be enabled and initialized.
- When objects are partially transparent (as indicated by their alpha value), their depth is not written into the depth buffer (as we can still see a portion of objects that are behind them).
- A common way to blend color and alpha with their existing values is as follows:

$$C = C_a\alpha_a + (1-\alpha_a)C_b\alpha_b$$

$$\alpha = \alpha_a + (1-\alpha_a)\alpha_b$$

- When a scene contains partially transparent objects, the final color of a pixel can differ depending on the order in which triangles are considered. Can you construct an example where this is true?
- This problem can be addressed to some extent. The wikipedia entry on alpha compositing contains a wealth of interesting additional detail: http://en.wikipedia.org/wiki/Alpha_compositing
- An alternative to using the depth buffer is to employ the Painter's Algorithm, which involves sorting objects back to front, splitting them if they intersect so that a consistent depth ordering can be obtained. Objects are then rasterized back to front, so that closer objects may "overpaint" objects that are behind them.
- There are other buffers in OpenGL whose properties you can investigate, including the accumulation and the stencil buffers. We will likely mention these buffers later in the class.