CS380: Introduction to Computer Graphics

Depth

Chapter 11

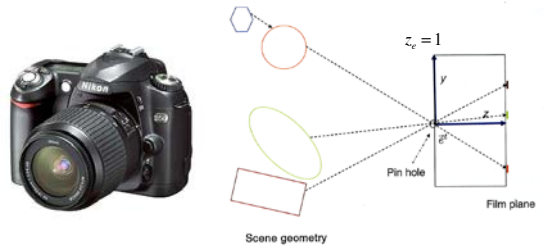
Min H. Kim

KAIST School of Computing

Projection

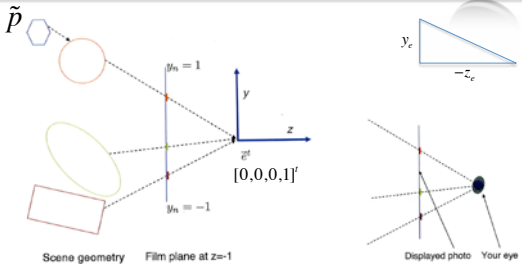
SUMMARY

Pinhole camera model



- Only rays of light that pass through this point reach the film plane and have their intensity recorded on film. The image is recorded at a film plane placed at, say, $z_e = 1$

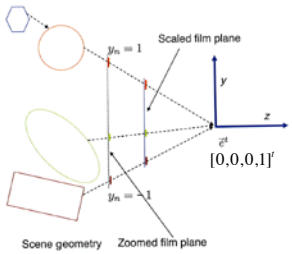
Normalized device coordinates



- Canonical square space:

$$x_n = -\frac{x_e}{z_e},\ y_n = -\frac{y_e}{z_e} \quad \Rightarrow \quad \begin{bmatrix} x_n w_n \\ y_n w_n \\ - \\ w_n \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ - \\ w_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

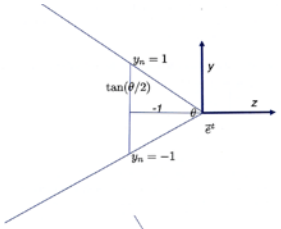
Scale factor *n*



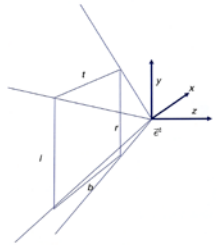
- Controlling aspect ratio of film space

$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ - \\ w_n \end{bmatrix} = \begin{bmatrix} -n & 0 & 0 & 0 \\ 0 & -n & 0 & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x_n w_n \\ y_n w_n \\ - \\ w_n \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

Frustum: Eye coor. → NDC



$$\begin{bmatrix} \frac{1}{\alpha \tan\left(\frac{\theta}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta}{2}\right)} & 0 & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

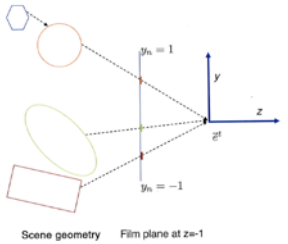


$$\begin{bmatrix} -\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & -\frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Chapter 11

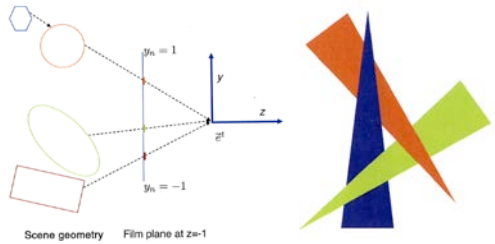
DEPTH

Visibility



- In the real world, opaque objects block light.
- We need to model this computationally.
- One idea is to render back to front and use overwriting
  - This will have problem with visibility cycles.

Visibility



- We could explicitly store everything hit along a ray and then compute the closest.
  - Make sense in a ray tracing setting, where we are working one pixel per ray at time, but not for OpenGL, where we are working one triangle at a time.

Z-buffer

- We will use z-buffer
- Triangle are drawn in any order
- Each pixel in frame buffer stores 'depth' value of closest geometry observed so far.
- When a new triangle tries to set the color of a pixel, we first compare its depth to the value stored in the z-buffer.
- Only if the observed point in this triangle is closer, we overwrite the color and depth values of this pixel.

Z-buffer

- This is done per-pixel, so there is no cycle problems.
- There are optimizations, where z-testing is done, before the fragment shading is done.

Other uses of visibility calculations

- Visibility to a light source is useful for shadows.
  - We will talk about shadow mapping later.
  - We will do shadow calculations in a ray tracer.
- Visibility computation can also be used to speed up the rendering process.
  - If we know that some object is occluded from the camera, then we don't have to render the object in the first place.
  - We can use a conservative test.

Basic mathematical model

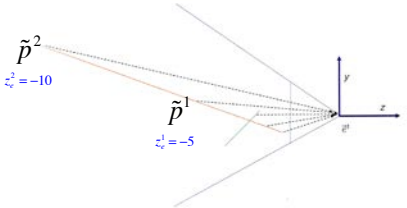
- For every point, we define its $[x_n, y_n, z_n]^t$ coordinates, using the following matrix expression:

$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} s_x & 0 & -c_x & 0 \\ 0 & s_y & -c_y & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

- We now also have the value $z_n = \frac{-1}{z_e}$
- Our plan is to use this z_n value to do depth comparison in our z-buffer.

Correct ordering

- Given two points $\tilde{p}^1$ and $\tilde{p}^2$ with eye coordinates $[x_e^1, y_e^1, z_e^1, 1]^t$ and $[x_e^2, y_e^2, z_e^2, 1]^t$.
- Suppose that they both are in front of the eye, i.e., $z_e^1 < 0$ and $z_e^2 < 0$.
- And suppose that $\tilde{p}^1$ is closer to the eye than $\tilde{p}^2$, that is $z_e^2 < z_e^1$
- Then $-\frac{1}{z_e^2} < -\frac{1}{z_e^1}$, meaning $z_e^2 < z_e^1$

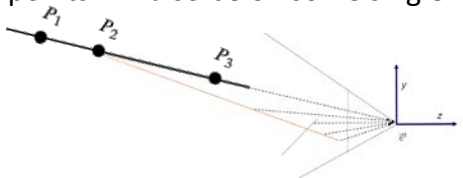


Projective transform

- We can now think of the process of taking points (given by eye coordinates) to points (given by normalized device coordinates) as an honest-to-goodness 3D geometric transformation.
- This kind of transformation is generally neither linear nor affine, but is something called a *3D projective transformation.*
- Projective transformation preserve co-linearity and co-planarity of points.

Co-linearity of points

- If three or more points are on a single line, the transformed points will also be on some single line.



- Three points $\mathbf{x}_i = [x_i, y_i, z_i, 1]$ for $i = 1,2,3$

$$x_2 - x_1 : y_2 - y_1 : z_2 - z_1 = x_3 - x_1 : y_3 - y_1 : z_3 - z_1$$

$$\left|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_1 - \mathbf{p}_3)\right| = 0$$

Co-planarity of points

- Analogous to an affine function *v* in the variables *x* and *y:* $v(x,y) = ax + by + c$
- We can rewrite it in matrix form:

$$v = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

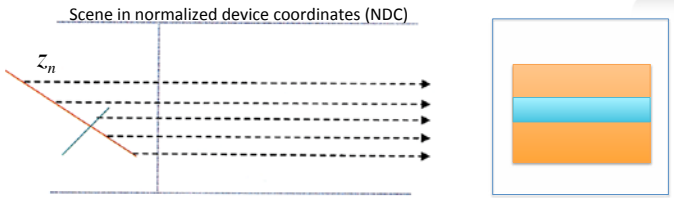
- A 3D projective transform maps planar objects in 3D to planar objects in 3D.
- As such, given a triangle in 3D and a selected eye frame and projective matrix, the value z_n at a point on a 3D triangle is an affine function of the (x_n, y_n) : $z_n = ax_n + by_n + c$

Co-planarity of points

- If we are given v_i, the values of v for three (non-collinear) points in the *(x,y)* plane, say the vertices of a triangle, this determine *v* over the entire plane.
- In this case, we say that v is the *linear interpolant* of the values at the three vertices.
- The process of evaluating the linear interpolant of three vertices is called *linear interpolation.*

$$\begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} =: \begin{bmatrix} a & b & c \end{bmatrix} M$$

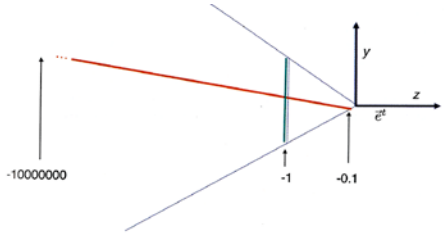
Co-planarity of points



- Note that distances are not preserved by a projective transform.
- Evenly spaced pixel on the film do not correspond to evenly spaced points on the geometry in eye space.
- Meanwhile, such *evenly spaced pixels* correspond with evenly spaced points in *normalized device coordinates.*

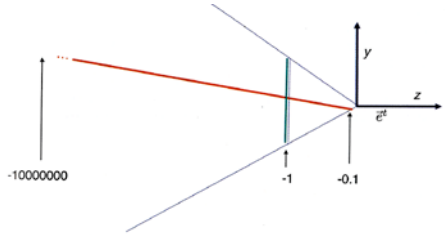
z_n interpolation is right

- Preservation of coplanarity: for points on a fixed triangle, we will have $z_n = ax_n + by_n + c$, for some fixed a, b and c
- Thus, the correct z_n value for a point can be computed using linear interpolation over the 2D image domain as long as we know its value at the three vertices of the triangle.



z_e interpolation is wrong

- Linear interpolation of z_e values over the screen would produce wrong answer.
- All the blue pixels would have interpolated value of -1. On the orange triangle z_e value would become less than -1 almost immediately.
- The entire image would become blue!!!



Numerics

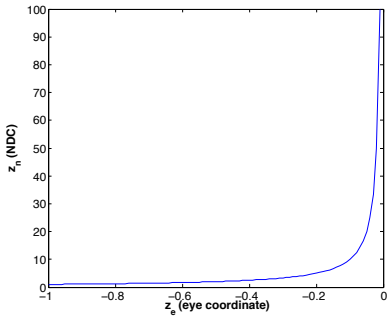
- There can be numerical difficulties when computing z_n. As z_e goes towards zero, the z_n value diverges off towards infinity. $z_n = \frac{-1}{z_e}$
- Conversely, points very far from the eye have z_n values very close to zero. The z_n of two such far away points may be indistinguishable in a finite precision representation, and thus the z-buffer will be ineffective in distinguishing which is closer to the eye.

Numerics

- points very far from the eye have z_n values very close to zero

$$z_n = \frac{-1}{z_e}$$

```
ze=-1*[0.01:0.01:10];
zn=-1./ze;
plot(ze(1:100),zn(1:100))
```



Solution: near/far

- Solution: replacing the third row of the matrix with more general row $\begin{bmatrix} 0 & 0 & \alpha & \beta \end{bmatrix}$

$$\rightarrow \begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} s_x & 0 & -c_x & 0 \\ 0 & s_y & -c_y & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

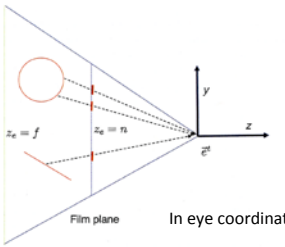
- It is easy to verify that if the value α and β are both positive, then the z-ordering of points (assuming they all have negative z_e values) is preserved under the projective transform.

Solution: near/far

- To set α and β, we first select depth value n and f called the *near* and *far* values (both negative), such that our main region of interest in the scene is sandwiched between $z_e = n$ and $z_e = f$
- Given these selections we set

$$\alpha = \frac{f+n}{f-n}$$

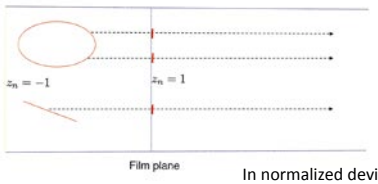
$$\beta = -\frac{2fn}{f-n}$$



In eye coordinates

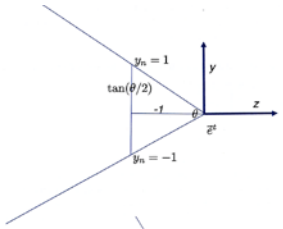
Solution: near/far

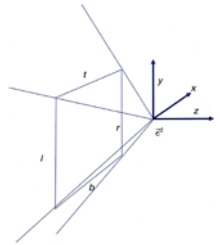
- We can verify now that any point with $z_e = f$ maps to a point with $z_n = -1$ and that a point with $z_e = n$ maps to a point with $z_n = 1$
- Any geometry not in this [near…far] range is clipped away by OpenGL and ignored.



In normalized device coordinates

Proj. Trans.: Eye coor. → NDC





$$\begin{bmatrix} \frac{1}{\alpha \tan\left(\frac{\theta}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & -\frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Codes

- In OpenGL, use the z-buffer is turned on with a call to glEnable(GL_DEPTH_TEST).
- We also need a call to glDepthFunc(GL_GREATER), since we are using a right handed coordinate system where 'more-negative' is 'farther from the eye'.
- In real life, you may see other conventions (for how to interpret n and f, some of the signs of the matrix, and the handedness of the ultimate z-test.