Viewer Optics

Sandro Spina
Computer Graphics and Simulation Group

Computer Science Department
University of Malta

Viewer Optics

- So far we’ve had a look at points, vectors and matrices ... Which are the primary tools used in CG.
- Optics also plays an important role. After all we need to render on screen something which is realistic to us.
- Hence, for example, we’ll need to use perspective because that is how we view things.
- We need to simulate our optics ... while projecting information of our 3D world scene onto a 2D space.

Projections

- A projection essentially provides a mechanism by which points in 3D space are mapped onto a 2D plane.

- We shall be deriving two types of projections commonly used in computer graphics, namely:
    - Orthographic Projection
    - Perspective Projection

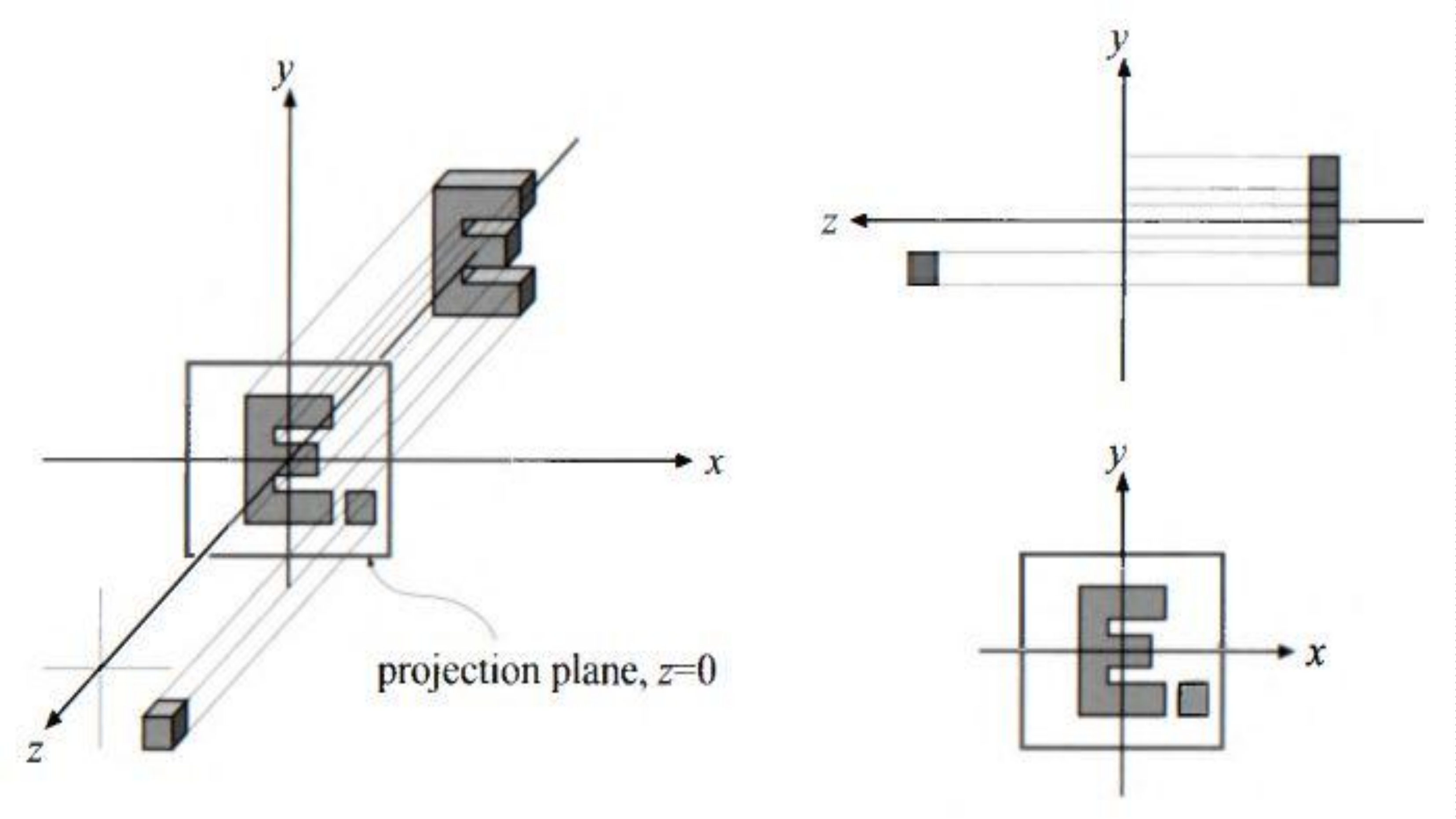
- A projection can also be used (and we'll be doing this) to map points from world space into a simple view volume.

- In CG before any rendering takes place, all relevant objects in the scene must be projected.

Projections as transforms

- All the transforms we have seen so far (scaling, translation, rotation, shearing) have left the fourth component, the w-component unaffected.
- Moreover, the bottom row in the 4 x 4 (homogenous notation) matrix has always been (0 0 0 1). Remember that we've used the first three elements of the 4^{th} column to represent translation.
- We'll see how the perspective projection will make use of this last row.

Orthographic Projection (visual)



Orthographic Projection

- An orthographic projection is one that maintains parallel lines (after projection).
- We can easily create a matrix which keeps the x, y components and zeros (flattens) the z value.
- The following matrix Po, carries out an orthographic projection on the plane z = 0.

$$P_o = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Orthographic Projection

- Clearly after performing the multiplication with P_o , we loose all the depth information.
- P_o is thus non-invertible, since its determinant = 0.
- Both +ve and –ve z-values are projected on the plane z = 0.
- It is usually useful (eg. for clipping purposes) to restrict the z, x and y values to a certain interval (a unit volume) ... This is done using planes which define a volume in world space.
- Another transformation matrix is utilised to carry out this task.

Orthographic Projection

- Orthographic projection (as a matrix) can also be expressed in terms of the six-tuple (l, r, b, t, n, f).

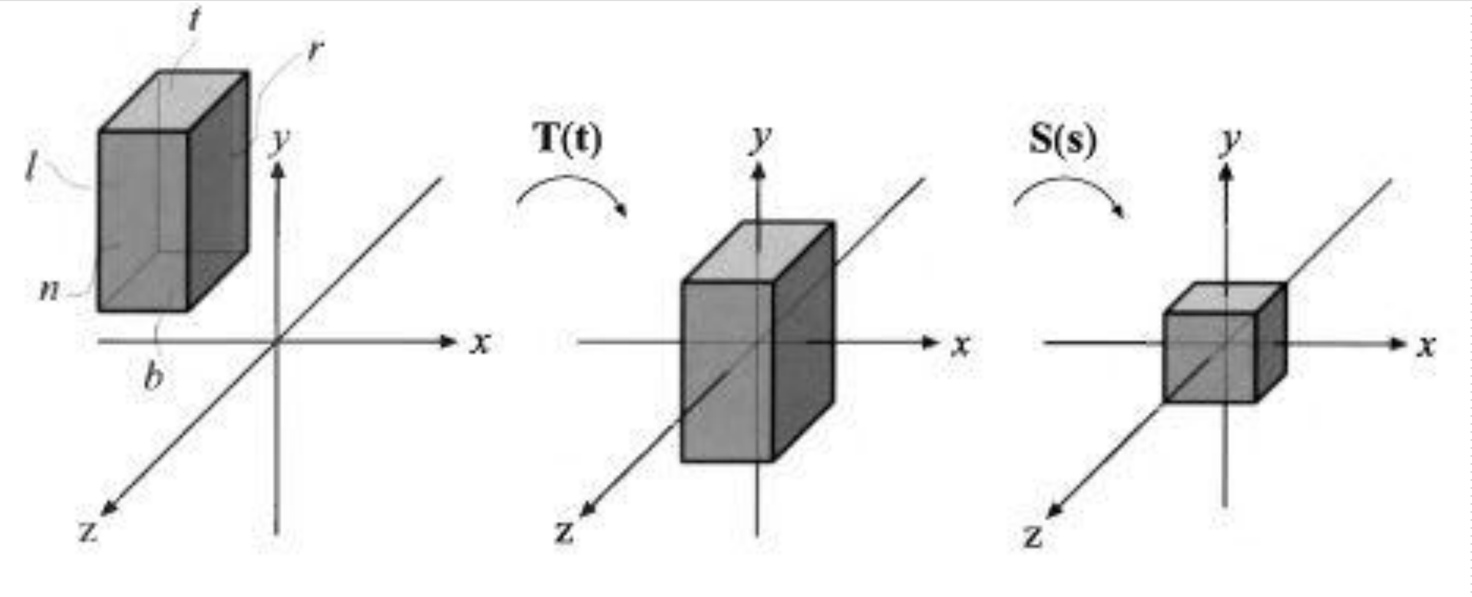
- l = Left plane
- r = Right Plane
- b = Bottom Plane
- t = Top Plane
- n = Near Plane
- f = Far Plane

- This matrix scales and translates the volume defined by these planes with the minimum corner = (l, b, n) and the maximum corner = (r, t, f) into an AABB centred around the origin.
- This cube (AABB) is referred to as the *Canonical View Volume*

Orthographic Projection

- The coordinates in this canonical view volume are referred to as the *normalised device coordinates*.

- Whatever geometry lies within this view volume will be rendered on screen ... The rest is clipped off.



Orthographic Projection

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

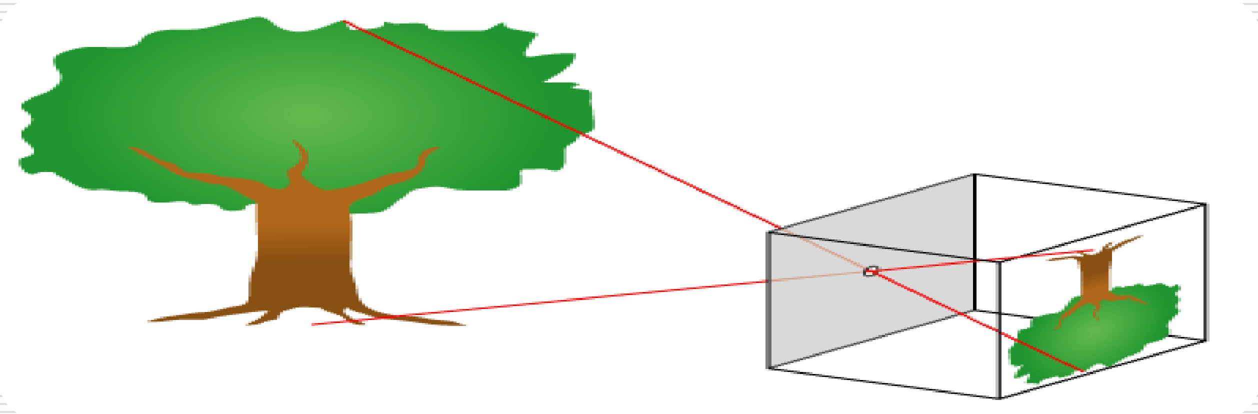
- Note that this matrix P_o is invertible …
- $(P_o)^{-1}$ would be equal to T(-t) S((r-l)/2, (t-b)/2, (f-n)/2)
- Note that since in the Canonical View Volume of DirectX z spans from 0 to 1 the matrix above would need to change slightly.

The Pinhole Camera (i)

- When we make use of our photographic equipment we are carrying out a projection which depends on the lens used.
- The simplest (earliest) form of camera is the pinhole camera which is useful in order to understand an important projection concept – perspective.
- We've seen that when using an orthographic projection, the projectors used are all parallel to each other. With perspective projection, projectors (rays of light) intersect at a point known as the centre of projection.
- Using a pinhole camera we can easily visualise this point, which stands in front of the projection plane.

The Pinhole Camera (ii)

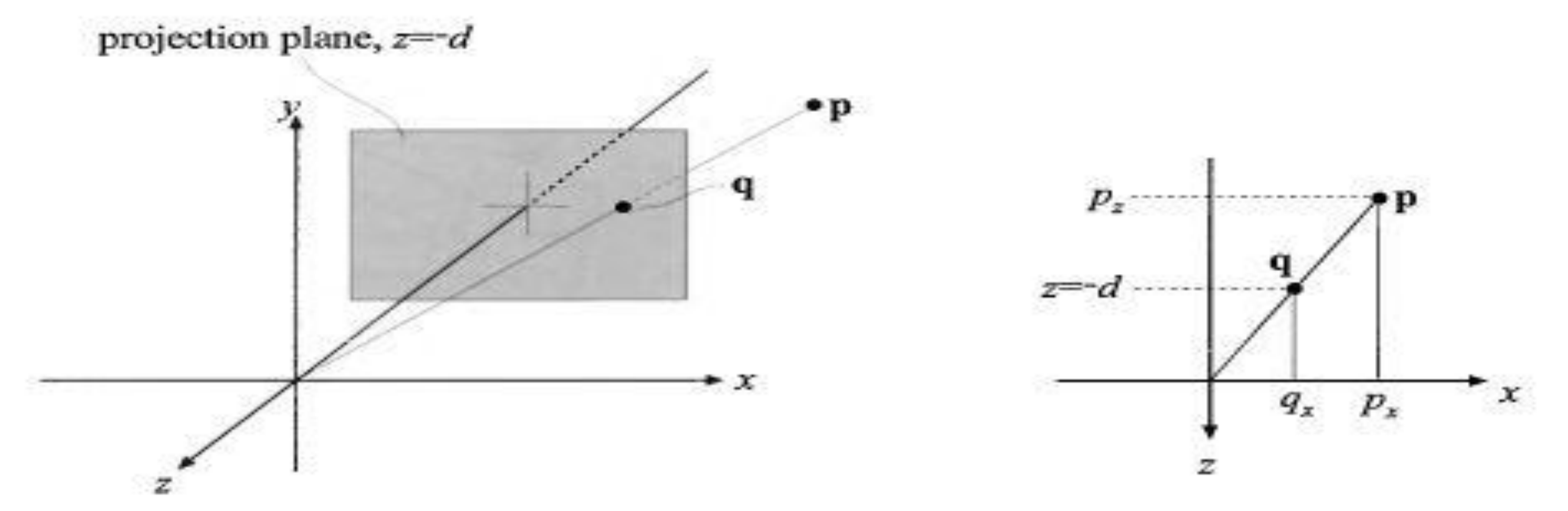
- A pinhole camera is a box with a tiny hole on one end.
- Rays of light enter the pinhole camera from this hole (Centre of Projection) and then hit the opposite end of the box (the projection plane).
- The image formed is inverted, since the rays of light (travelling in a 'straight line') cross as they meet at the pinhole (COP)



Perspective Projection (i)

- Inspired by the pinhole camera we can derive a much more useful (for us!!) projection which is the perspective projection.
- As opposed to the orthographic projection, parallel lines are not parallel after projection; they may actually converge to a single point at their extreme. (Think of a railway track going into the screen)
- Perspective projection matches more closely how we perceive the things around us … the farther away they are the smaller we see them. Perspective foreshortening.
- We shall start this off by a derivation of a matrix which projects (using perspective) vertices on a near plane.

Perspective Projection (ii)



- First assume that our viewpoint (camera) is located at the origin and that we want to project **p** (above) onto the projection place z = -d, d > 0
- The new point will be q = (q_x, q_y, -d)

Perspective Projection (iii)

- Using the similar triangles in the previous diagram one can infer that (for the x-component)

$$\frac{q_x}{p_x} = \frac{-d}{p_z} \qquad \leftrightarrow \qquad q_x = -d\frac{p_x}{p_z}$$

- Using a similar derivation we also get the value of q_y

$$\frac{q_y}{p_y} = \frac{-d}{p_z} \qquad \leftrightarrow \qquad q_y = -d\frac{p_y}{p_z}$$

- Clearly q_z = -d, hence the projection of p on plane –d is given by the coordinates (-d p_x/p_y, -d p_y/p_z, -d)

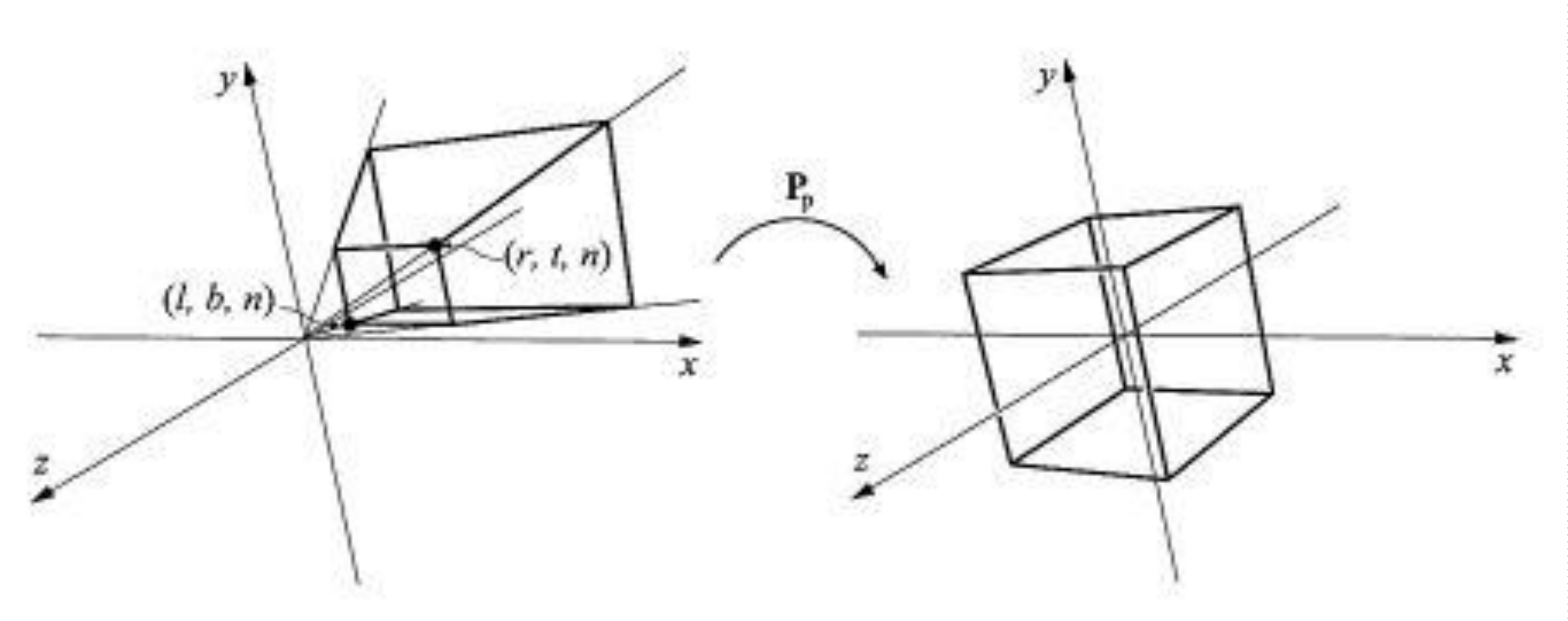
Perspective Projection (iv)

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ -p_z/d \end{pmatrix} \rightarrow \begin{pmatrix} -dp_x/p_z \\ -dp_y/p_z \\ -d \\ 1 \end{pmatrix}$$

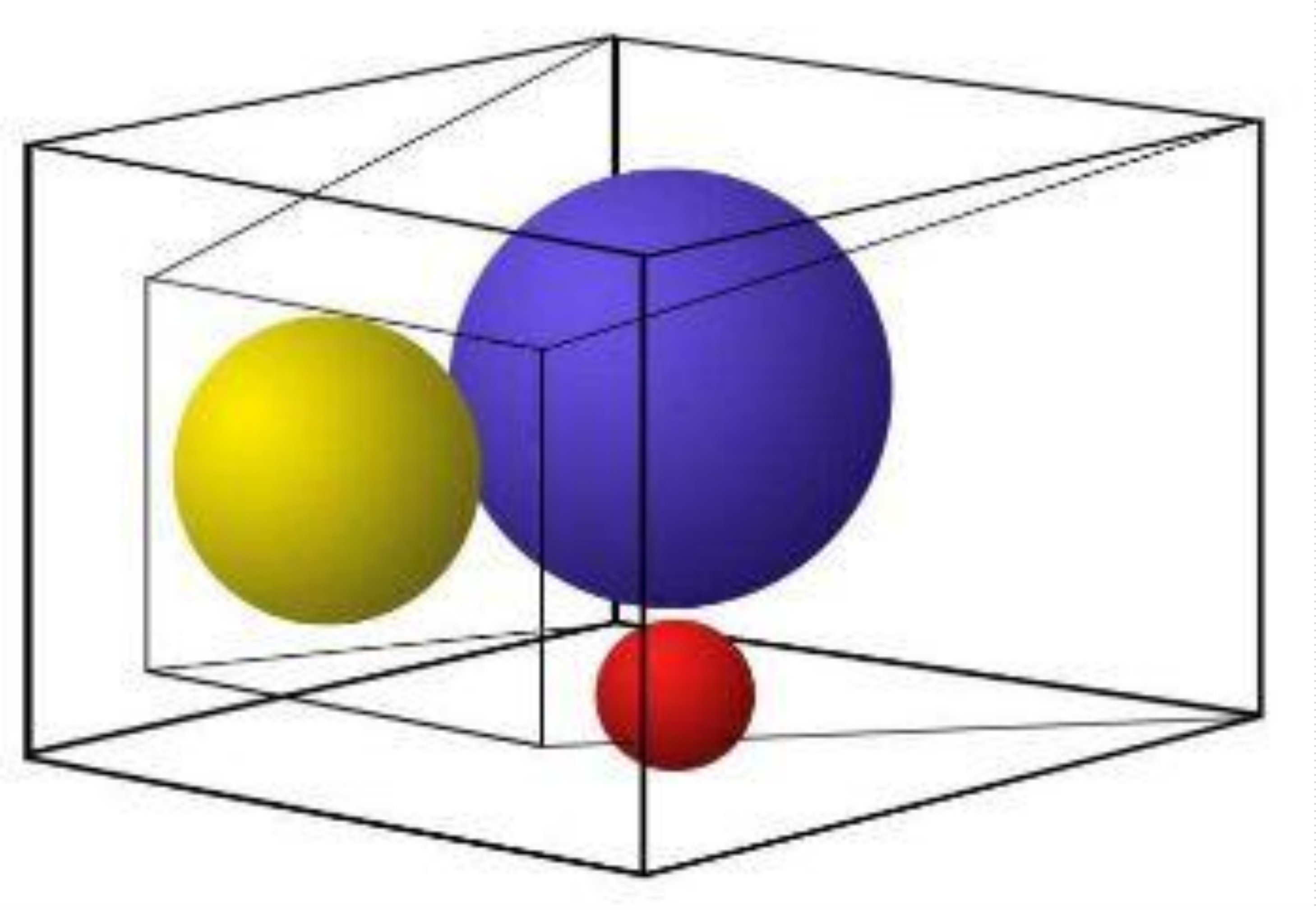
- Assume P = (p_x, p_y, p_z,1) … then multiplying by this projection matrix gives us the required value Q on the projection plane after dividing by the w-component.
- The problem (similar to the one we had with orthographic projection) is that the matrix does not have an inverse … hence we lose our z-coordinate.

Perspective Projection (v)

- For the perspective projection we shall also derive a perspective transform matrix which transforms the view frustum into the canonical view volume.



Perspective Projection Picture



Perspective Projection (vi)

- Note that (as seen in the prev diagram) the rectangle at z = n (near plane) has the minimum corner at (l,b,n) and the maximum corner at (r,t,n)
- These parameters (l,r,b,t,n,f) determine the view frustum of the camera.
- They also determine the horizontal (angle between l and r) and vertical (angle between b and t) fields of view.
- When using a narrower field of view (equivalent to zooming in with your photographic camera) the perpective effect is lessened.
- However increasing the FOV (for eg using a fish eye lens) will make objects appear distorted.

Perspective Projection (vii)

- The perspective transform matrix that transforms the frustum into a unit cube is given below:

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

- After applying this transform to a point we get the projected point by dividing by the w-component. This gives us the NDC in the view volume.

- Note that the matrix always sees that the projected point in the view volume is assigned to +1 when z = f and to -1 when z = n. We'll check this out in the next slide.

Perspective Projection (viii)

- Let us take points p = (1, 0, 2), q = (3, 0, 7) and r = (0, 0, 1) and the near plane to be z=1 and far plane to be z = 7.

- Multiplying p by P_p we get

$$\begin{pmatrix} \frac{2n}{r-l} - 2(\frac{r+l}{r-l}) \\ -2(\frac{t+b}{t-b}) \\ 2\left(\frac{8}{6}\right) - \frac{7}{6} \\ 2 \end{pmatrix} \text{»} \begin{pmatrix} \frac{2n}{r-l} - 2(\frac{r+l}{r-l}) \\ -2(\frac{t+b}{t-b}) \\ 0.333 \\ 2 \end{pmatrix} \text{»} \begin{pmatrix} \frac{2n}{r-l} - 2(\frac{r+l}{r-l}) \\ -2(\frac{t+b}{t-b}) \\ 0.166 \\ 1 \end{pmatrix}$$

- Multiplying q by P_p we get

$$\begin{pmatrix} \frac{2n}{r-l} - 2(\frac{r+l}{r-l}) \\ -2(\frac{t+b}{t-b}) \\ 7\left(\frac{8}{6}\right) - \frac{14}{6} \\ 7 \end{pmatrix} \text{»} \begin{pmatrix} \frac{2n}{r-l} - 2(\frac{r+l}{r-l}) \\ -2(\frac{t+b}{t-b}) \\ 7 \\ 7 \end{pmatrix} \text{»} \begin{pmatrix} \frac{2n}{r-l} - 2(\frac{r+l}{r-l}) \\ -2(\frac{t+b}{t-b}) \\ 1 \\ 1 \end{pmatrix}$$

- Multiplying r by P_p we get

$$\begin{pmatrix} \frac{2n}{r-l} - 2(\frac{r+l}{r-l}) \\ -2(\frac{t+b}{t-b}) \\ \left(\frac{8}{6}\right) - \frac{14}{6} \\ 1 \end{pmatrix} \text{»} \begin{pmatrix} \frac{2n}{r-l} - 2(\frac{r+l}{r-l}) \\ -2(\frac{t+b}{t-b}) \\ -1 \\ 1 \end{pmatrix} \text{»} \begin{pmatrix} \frac{2n}{r-l} - 2(\frac{r+l}{r-l}) \\ -2(\frac{t+b}{t-b}) \\ -1 \\ 1 \end{pmatrix}$$

Creating a Camera in DirectX (i)

- Given our knowledge so far we know that we need at least two matrices to setup our camera.
  - Matrix projection;
  - Matrix view;

- An InitialiseCamera() method can be setup and used to initialise your camera before starting to load content.

- To set up the projection matrix we need to first figure out the aspect ratio (width/height) of the viewport and then call the CreatePerspectiveFieldOfView() method.

Creating a Camera in DirectX (ii)

- Float aspectRatio = (float)graphics.GraphicsDevice.Viewport.Width / (float)graphics.GraphicsDevice.Viewport.Height;
- Matrix.createPerspectiveFieldOfView(MathHelper.PiOver4, aspectRatio, 0.0001f, 1000.0f, out projection)
- The variable projection now stores the required matrix to transform world space vertices into NDC.
- Once projection is done we can then focus on the view matrix.

Creating a Camera in DirectX (iii)

- The projection matrix defines the 'lens' used with the camera …. Whereas the view matrix will define the location and orientation of the camera itself.
- The view is what the camera sees.
- XNA will again help us out in coming up with the matrix … working out the necessary math.
- We'll use a Matrix helper method, Matrix.CreateLookAt() in order to specify the view matrix.

Creating a Camera in DirectX (iv)

- Vector3 cameraPosition = new Vector3(0.0f, 0.0f, 3.0f);   //backwards from the origin by 3 units.
- Vector3 cameraTarget = Vector3.Zero;
- Vector3 cameraUpVector = Vector3.Up;

- Matrix.CreateLookAt(ref cameraPosition, ref cameraTarget, ref cameraUpVector, out view);

- Vector3.Up = (0, 1, 0)

- Note that we are specifically passing everything by reference (not by value) to gain performance.

Creating a Camera in DirectX (v)

- Even though not part of the camera setup we clearly also need to setup the world matrix.
- Matrix world = Matrix.Identity; //recall opengl
- The above means that whatever objects I'm drawing now will neither be rotated, scaled or translated.
- Matrix.Identity sets the world matrix to the origin of the world. We now just need to add stuff !!

XNA Examples ...

http://farm4.static.flickr.com/3489/3293014
047_51c78a5674.jpg

- Load the XNA example which demos how to create a camera component in XNA.