Computer Graphics

-Transformations and Projections-

Oliver Bimber

Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

NEXT ICG LAB TALK:
MARCH 15, 2016, 4:30PM

Prof. Cagatay Turkay

City University London

Interactive Visual Analysis to Aid Data-informed Analytical Problem Solving

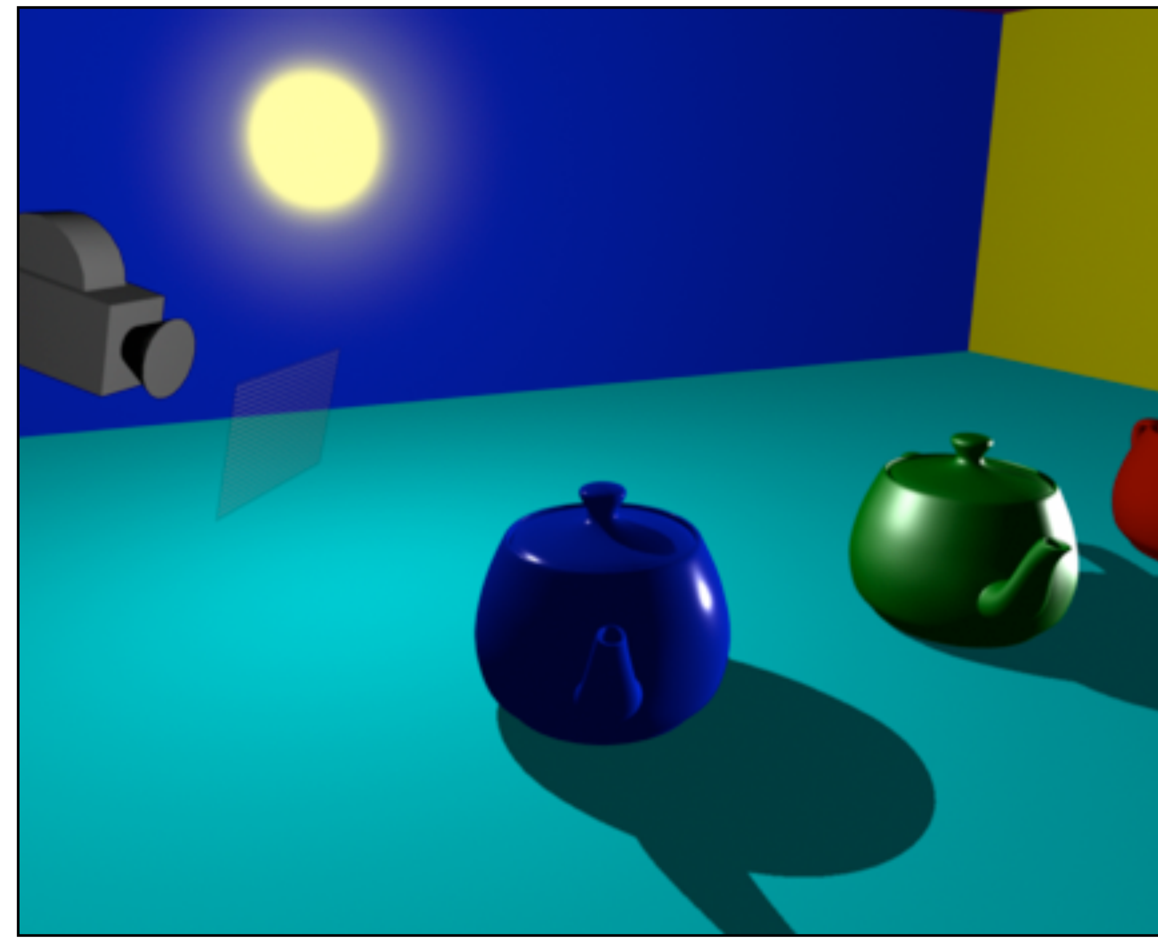
Computer Science Building (SP3)

Room SP2 054

For more information about our talks visit **http://www.cg.jku.at/talks/invited**

Graphical Scenes

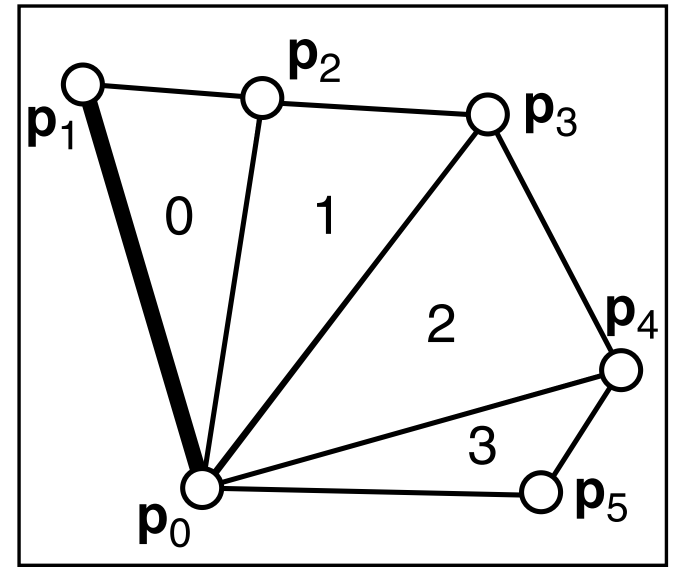
- In computer graphics (CG), a scenery has to be defined (e.g., modeled) before it can be rendered (i.e., computing a picture of it)
- This includes the definition of scene geometry (e.g., surfaces), material properties, light sources, and a camera / viewpoint from which the scene has to be rendered
- Since camera, scene elements, light sources, etc. can dynamically move (either interactively or as part of an animation), we need to continuously re-compute their new relations for rendering
- So, how are scenes normally defined and how do we compute (2D) pixel positions in a rendered image for given (3D) scenery and camera?

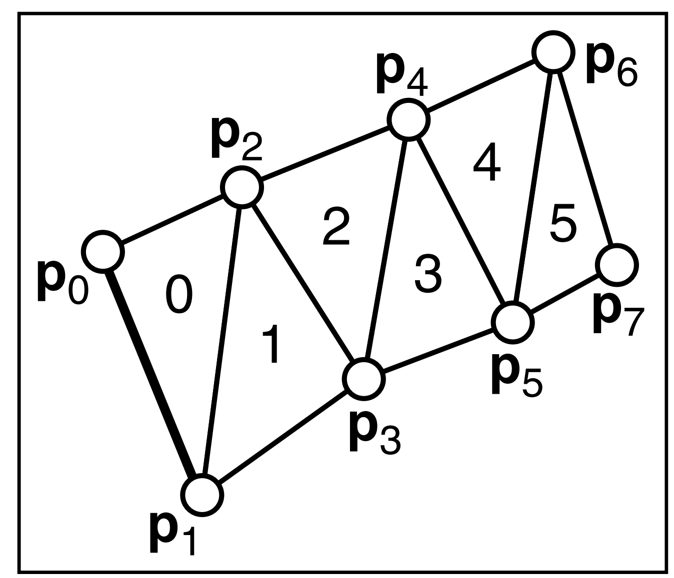


3D scene modeling in CG

Geometric Primitives

- In CG, 2D and 3D surfaces are usually formed through different meshes of triangles
- These meshes are defined by points (called vertices) and their connections (called edges)
- Vertices and triangles can be characterized through different properties, such as positions, normal vectors, color, texture coordinates, etc.
- How are these surfaces transformed?





Basic Transforms in 2D

- 2x2 matrices can be used to change the components of 2D vectors
- This includes rotations, scales, and shears - but no translations (we'll see later)

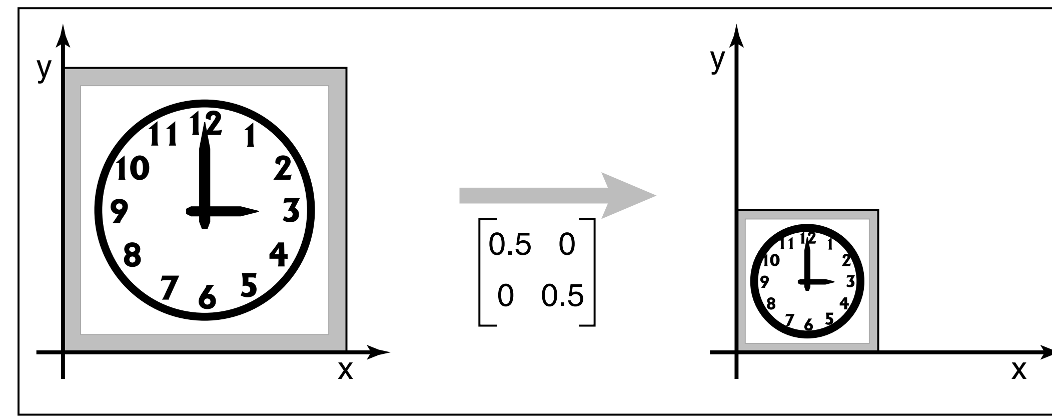
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

Basic Transforms in 2D

- 2x2 matrices can be used to change the components of 2D vectors

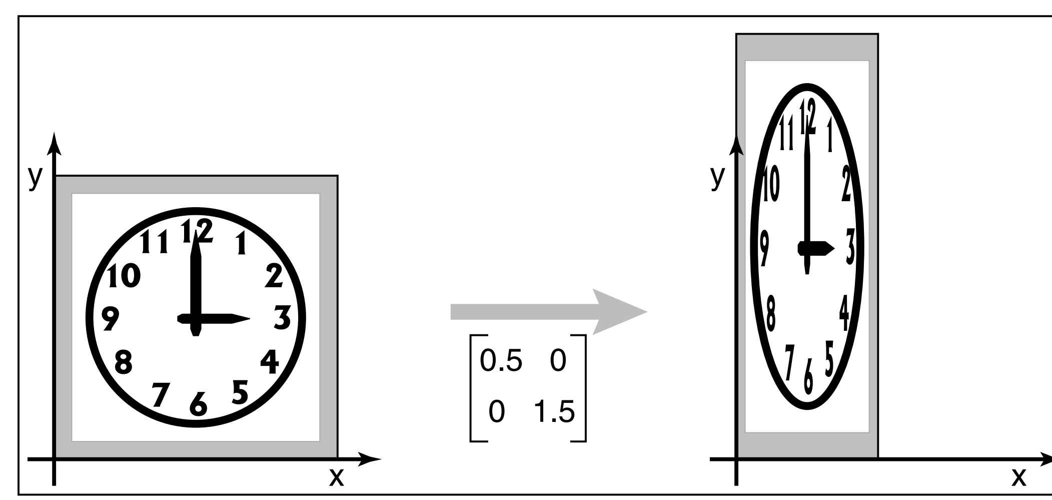
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

- This includes rotations, scales, and shears - but no translations (we'll see later)



- For scalings, s_x and s_y are the scaling factors in x and y directions

$$scale(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

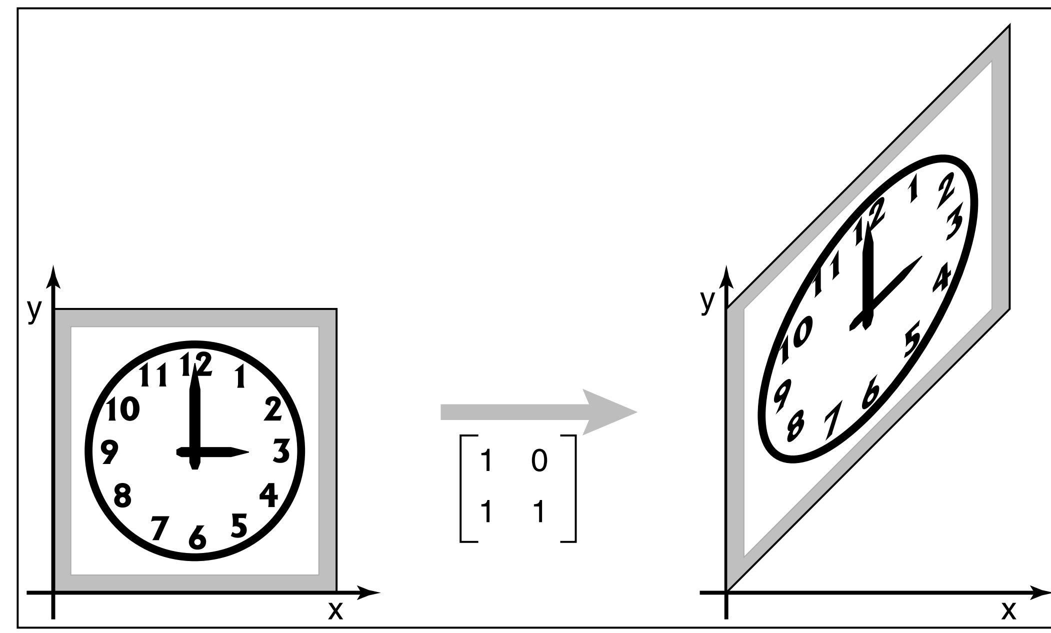


Basic Transforms in 2D

- 2x2 matrices can be used to change the components of 2D vectors

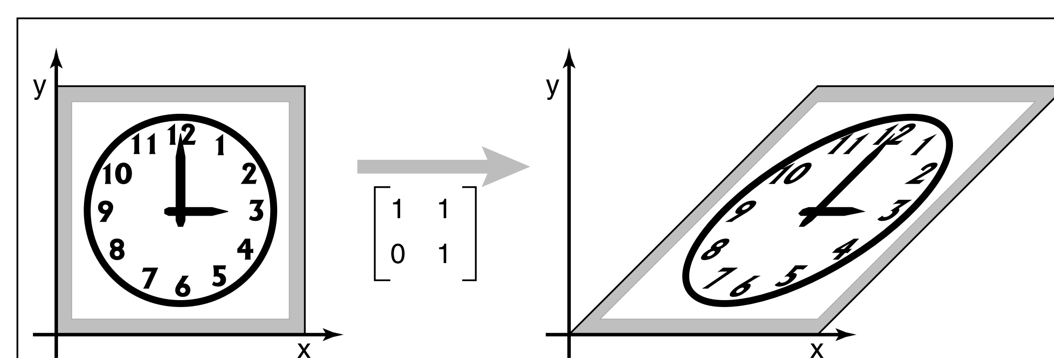
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

- This includes rotations, scales, and shears - but no translations (we'll see later)



- Shear transformations warp objects while horizontal and vertical edges remain parallel

$$shear_x(s) = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \quad shear_y(s) = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}$$

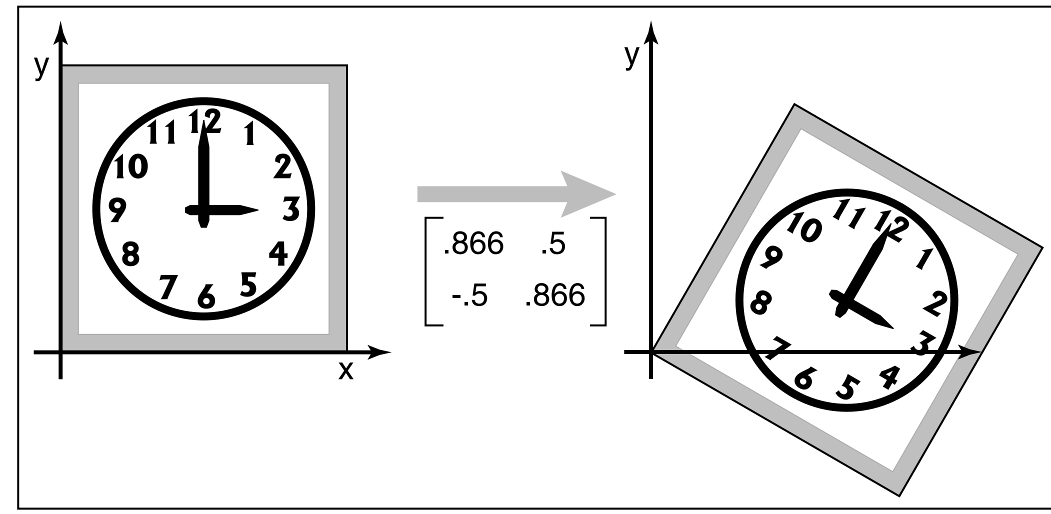


Basic Transforms in 2D

- 2x2 matrices can be used to change the components of 2D vectors

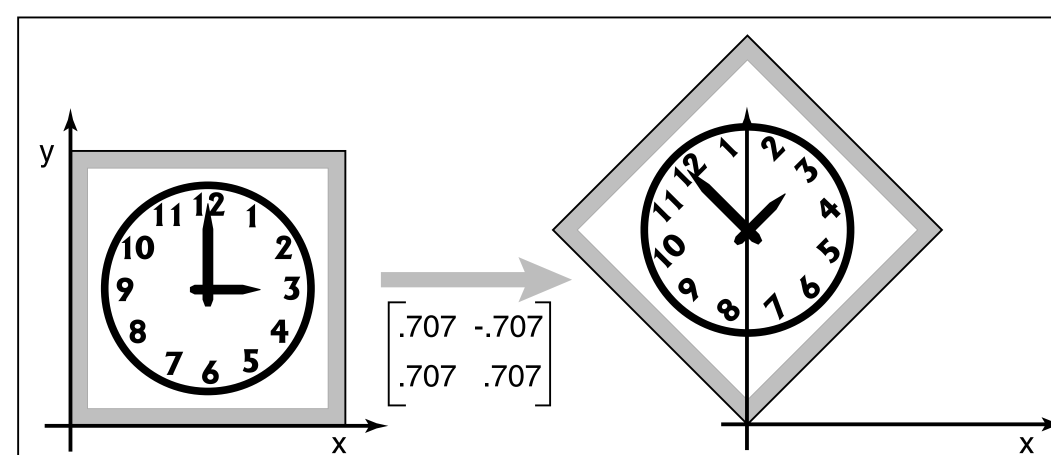
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

- This includes rotations, scales, and shears - but no translations (we'll see later)



- Rotations rotate vectors by angle (phi) around the origin (x,y=0,0)

$$rotate(\phi) = \begin{bmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix}$$

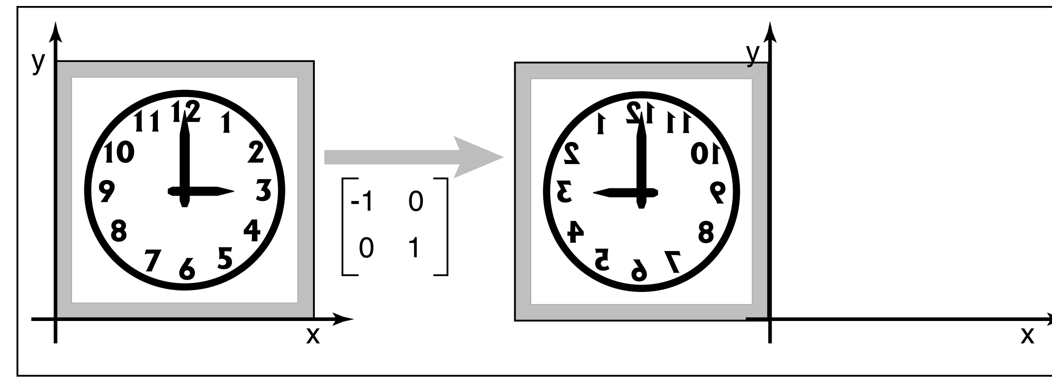


Basic Transforms in 2D

- 2x2 matrices can be used to change the components of 2D vectors

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

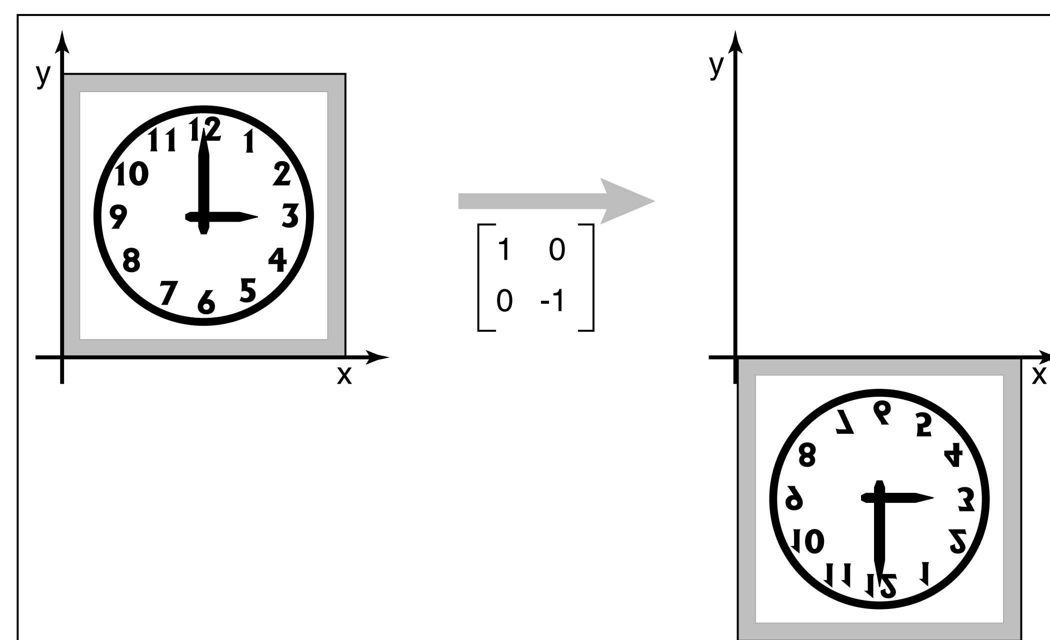
- This includes rotations, scales, and shears - but no translations (we'll see later)



- Reflections flip vectors around x- or y-axis

$$reflect_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad reflect_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

- How would you combine multiple transformations (say: a rotation, followed by a translation, followed by another rotation)?



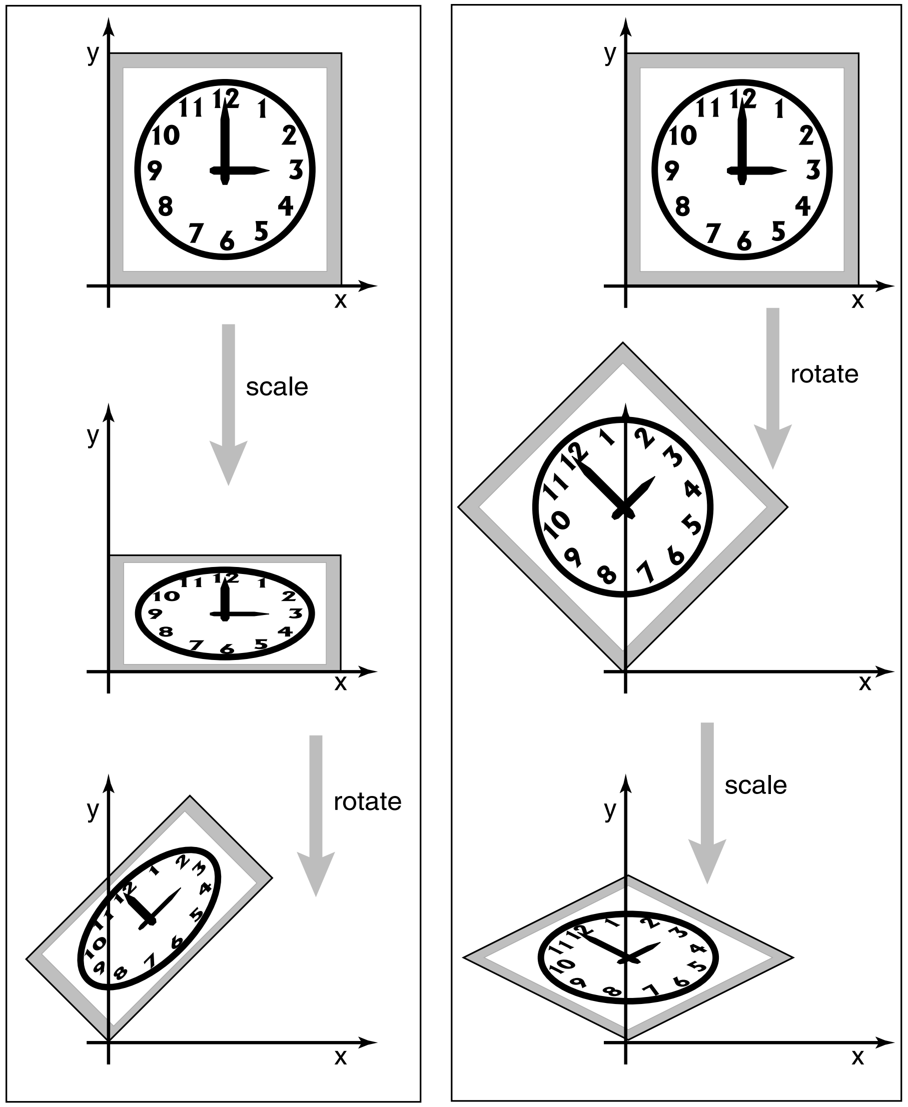
Compositions of 2D Transforms

- Applying two transform matrices M_1 and M_2 in a sequence is equivalent to applying the product of both matrices
- This means that two (or more) transform matrices can be composed into one (M_3) through matrix multiplication (remember that matrix multiplication is associative)
- The order of the multiplications matters - the transformations are applied from the right side first
- Note: every composed matrix can be decomposed via singular value decomposition (SVD) into a product of rotation*scale*rotation
- What about translations?

$$v_2 = M_1 v_1, v_3 = M_2 v_2$$

$$v_3 = M_2(M_1 v_1) = (M_2 M_1) v_1$$

$$v_3 = M_3 v_1, M_3 = M_2 M_1$$



Translations in 2D

- They cannot be modeled with such a simple equation system (i.e., not with a 2x2 matrix)

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

- Solution: use a larger matrix (3x3)

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

- What about the other transformations now?

Translations in 2D

- They cannot be modeled with such a simple equation system (i.e., not with a 2x2 matrix)

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

- Solution: use a larger matrix (3x3)

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

- What about the other transformations now?
- Solution: extending previous (2x2) transformation matrix (rotation, scaling, shear, reflection) to 3x3 and multiplying it with 3x3 translation matrix

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

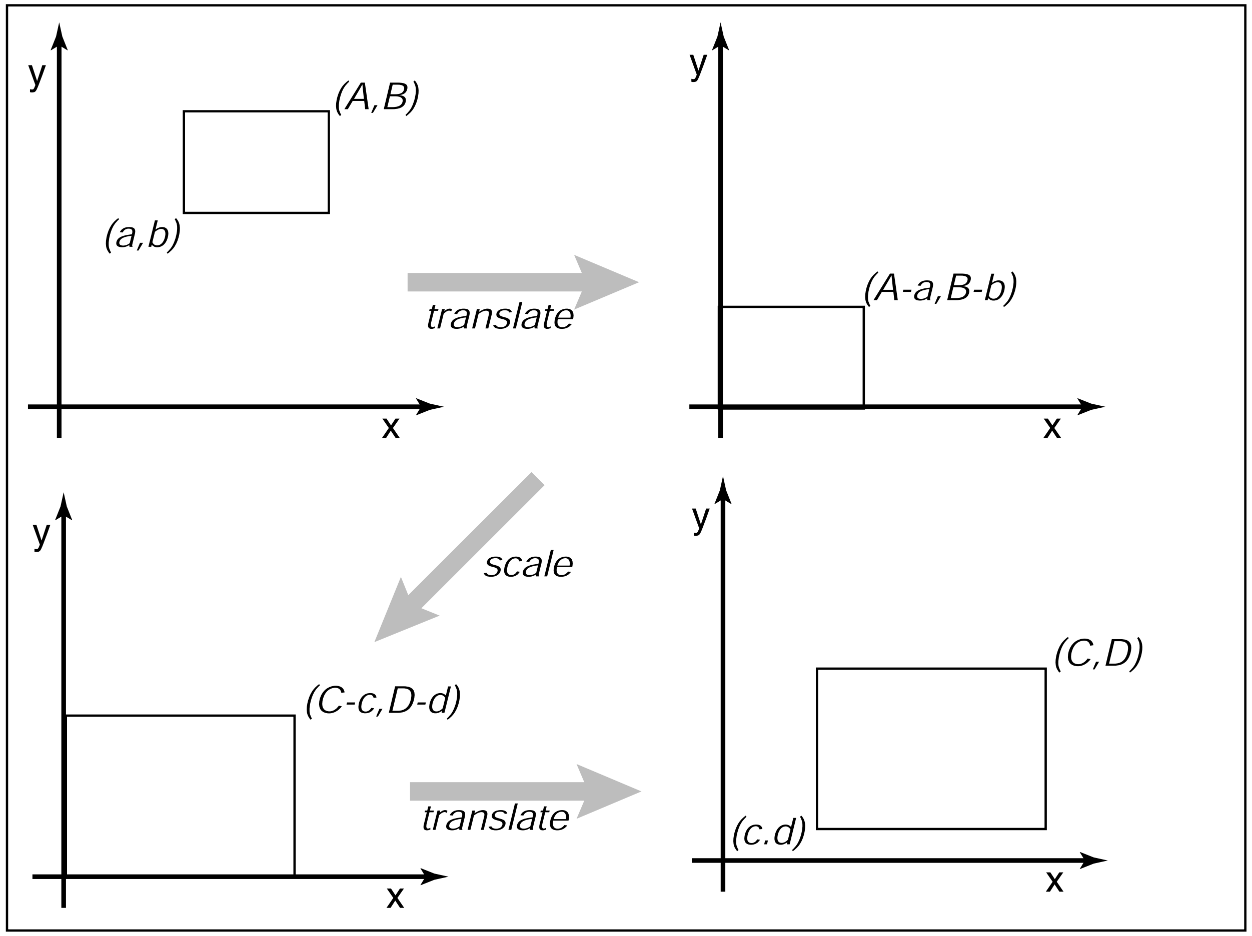
- Thus, all 2D transformations can be represented with a single 3x3 matrix

Note: so-called *rigid-body transforms* contain only rotations and translations!

Viewport Transforms

- Also known as window transforms or windowing transforms
- Transforms one 2D graphics window to another one
- Can be modeled as a simple composition of scale and translate transforms

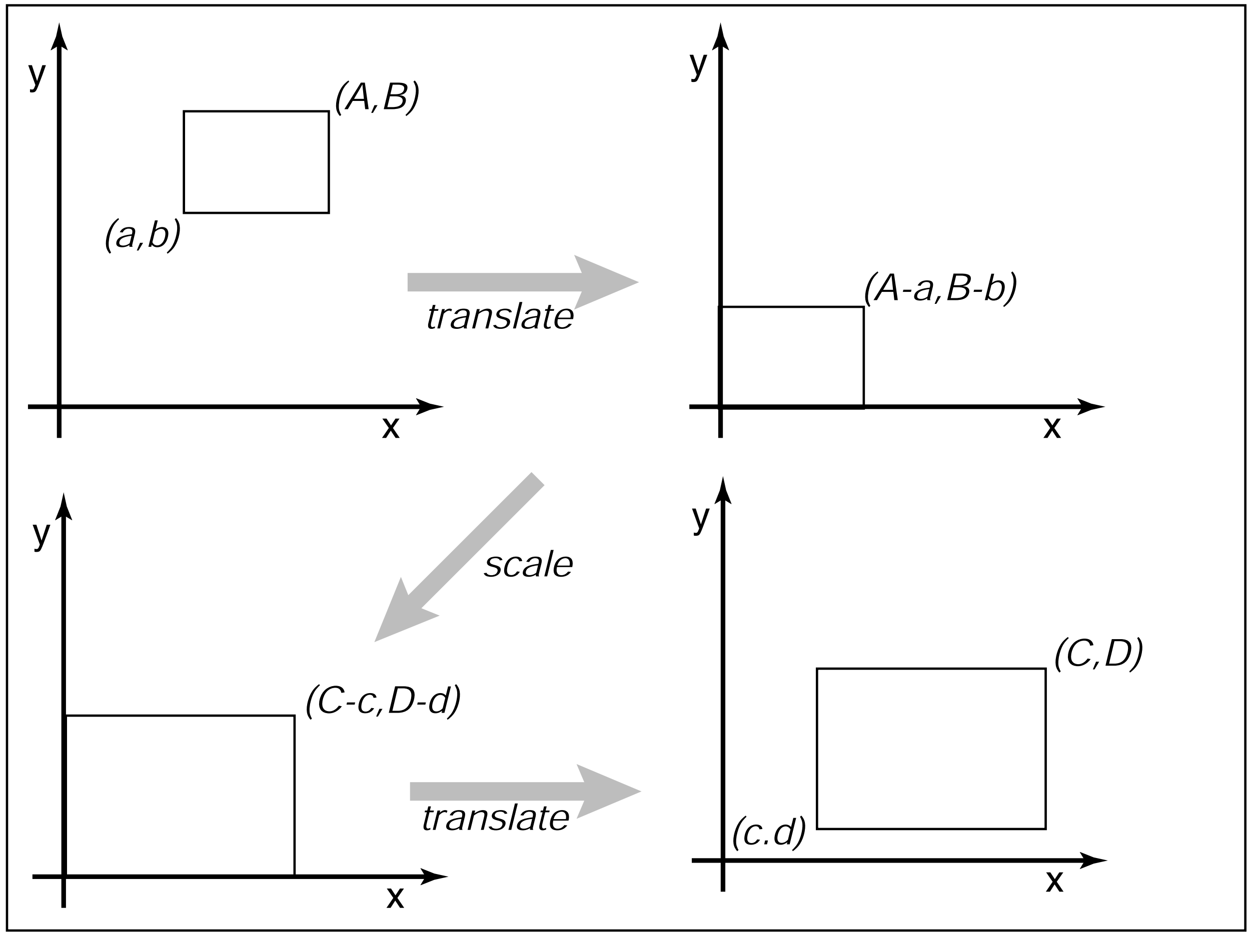
1. translate to origin
2. scale to new dimension
3. translate to new position



Viewport Transforms

- Also known as window transforms or windowing transforms
- Transforms one 2D graphics window to another one
- Can be modeled as a simple composition of scale and translate transforms

1. translate to origin
2. scale to new dimension
3. translate to new position



$$\begin{bmatrix} 1 & 0 & c \\ 0 & 1 & d \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{C-c}{A-a} & 0 & 0 \\ 0 & \frac{D-d}{B-b} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{C-c}{A-a} & 0 & \frac{cA-Ca}{A-a} \\ 0 & \frac{D-d}{B-b} & \frac{dB-Db}{B-b} \\ 0 & 0 & 1 \end{bmatrix}$$

multiply from right!

Basic Transforms in 3D

- Moving from 2D to 3D is straight forward
- Rotation, scale, shear, reflection can be modeled as 3x3 matrices
- Simple rotations around individual axes can be combined (again, trough multiplication)
- To include translations, the 3D transform matrix has to be extended to 4x4

$$scale(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

$$shear_x(s_y, s_z) = \begin{bmatrix} 1 & s_y & s_z \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

similar for y and z axes

$$rotate_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix}$$

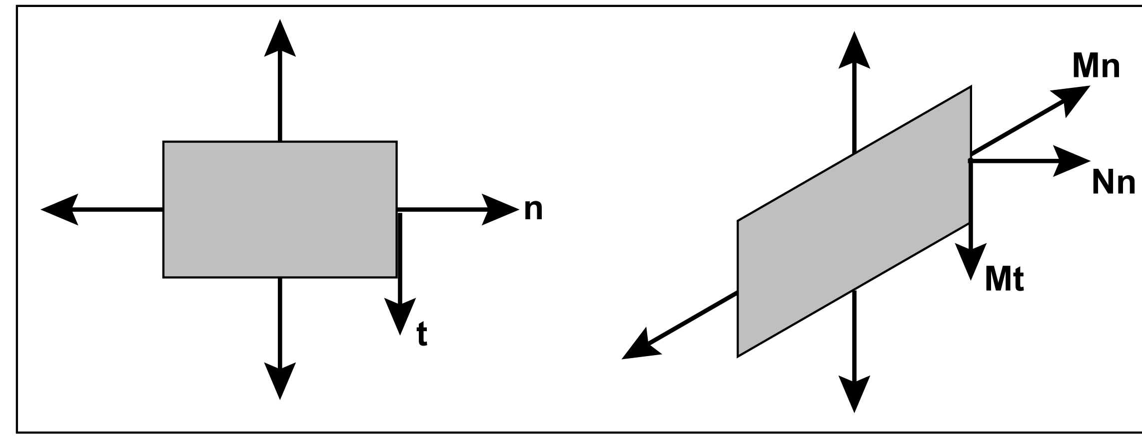
$$rotate_y(\phi) = \begin{bmatrix} cos(\phi) & 0 & sin(\phi) \\ 0 & 1 & 0 \\ -sin(\phi) & 0 & cos(\phi) \end{bmatrix}$$

$$rotate_z(\phi) = \begin{bmatrix} cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transforming Normal Vectors

- We said earlier, that normal vectors are properties of vertices (called vertex normals) and triangles (called surface normals)
- They are vectors that describe directions (e.g., perpendicular to a plane), and not positions
- They can not be transformed with our model because they might not remain perpendicular to their plane after transformation
- Any idea of how to support the correct transformation of direction vectors, such as normals?



Before the transformation, the normal vector (n) is perpendicular to the tangent vector (t)

After the transformation M (a shear as an example), Mn is no longer perpendicular to Mt - yet, Mt is still tangential and all vertices (position vectors) are transformed correctly - but we need Nn - and therefore need to find N

Transforming Normal Vectors

- To solve this, we remember that normal (n) and tangent vector (t) are perpendicular, and their dot product must be 0

$$n^T t = 0$$

Transforming Normal Vectors

- To solve this, we remember that normal (n) and tangent vector (t) are perpendicular, and their dot product must be 0

$$n^T t = 0$$

- This applies also to the transformed n and t (we need to find N)

$$(Nn)^T M t = 0$$

Transforming Normal Vectors

- To solve this, we remember that normal (n) and tangent vector (t) are perpendicular, and their dot product must be 0

$$n^T t = 0$$

- This applies also to the transformed n and t (we need to find N)

$$(Nn)^T M t = 0$$

- Multiplying with the identity (I) does not change the equation above

$$n^T t = n^T I t = n^T M^{-1} M t = 0$$
$$M^{-1} M = I$$

Transforming Normal Vectors

- To solve this, we remember that normal (n) and tangent vector (t) are perpendicular, and their dot product must be 0

$$n^T t = 0$$

- This applies also to the transformed n and t (we need to find N)

$$(Nn)^T Mt = 0$$

- Multiplying with the identity (I) does not change the equation above

$$n^T t = n^T I t = n^T M^{-1} M t = 0$$

$$M^{-1} M = I$$

- Rewriting this with parentheses makes it easy to see what N must be

$$(n^T M^{-1})(Mt) = 0$$

Transforming Normal Vectors

- To solve this, we remember that normal (n) and tangent vector (t) are perpendicular, and their dot product must be 0
- This applies also to the transformed n and t (we need to find N)
- Multiplying with the identity (I) does not change the equation above
- Rewriting this with parentheses makes it easy to see what N must be
- It must be the transposed, inverse of M

$$n^T t = 0$$

$$(Nn)^T Mt = 0$$

$$n^T t = n^T It = n^T M^{-1} Mt = 0$$

$$M^{-1}M = I$$

$$(n^T M^{-1})(Mt) = 0$$

$$(n^T M^{-1}) = (Nn)^T$$

$$N = (M^{-1})^T$$

Transforming Normal Vectors

- To solve this, we remember that normal (n) and tangent vector (t) are perpendicular, and their dot product must be 0
- This applies also to the transformed n and t (we need to find N)
- Multiplying with the identity (I) does not change the equation above
- Rewriting this with parentheses makes it easy to see what N must be
- It must be the transposed, inverse of M
- N can contain a translation - but normal vectors should not be translated (they are direction vectors - not position vectors)
- How can we cancel the translational part?

$$n^T t = 0$$

$$(Nn)^T M t = 0$$

$$n^T t = n^T I t = n^T M^{-1} M t = 0$$

$$M^{-1} M = I$$

$$(n^T M^{-1})(M t) = 0$$

$$(n^T M^{-1}) = (Nn)^T$$

$$N = (M^{-1})^T$$

Transforming Direction Vectors

- In contrast to position vectors (e.g., vertices), we don't want direction vectors (e.g., normals) to be translated
- This is easy to solve by simply setting the 4th coordinate to either 1 or 0 (which will enable or disable the translation component)
- The same is true for the 3rd coordinate in the 2D case
- This last coordinate is usually referred to as homogeneous coordinate
- Later, we will see that the homogeneous coordinate can have an arbitrary value

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \boxed{1} \end{bmatrix}$$

position vector

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \boxed{0} \end{bmatrix}$$

direction vector

Transforming Direction Vectors

- In contrast to position vectors (e.g., vertices), we don't want direction vectors (e.g., normals) to be translated
- This is easy to solve by simply setting the 4th coordinate to either 1 or 0 (which will enable or disable the translation component)
- The same is true for the 3rd coordinate in the 2D case
- This last coordinate is usually referred to as homogeneous coordinate
- Later, we will see that the homogeneous coordinate can have an arbitrary value
- Sometimes, it is necessary to compute the inverse of a transformation matrix - do you know how?

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \boxed{1} \end{bmatrix} \quad \text{position vector}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \boxed{0} \end{bmatrix} \quad \text{direction vector}$$

Inverse Transformations

- The inverse of each transformation matrix can be computed numerically
- But if we know the composition of the transformation, its inverse can easily be computed from the reverse composition of the inverse components (more efficient)
- For example:
  - the inverse of a scale with s is a scale with 1/s
  - the inverse of a translation about x,y,z is a translation about -x,-y,-z
  - the inverse of a rotation is its transpose
- Matrix compositions can be inverted by inverting their components (reverse order of multiplications)

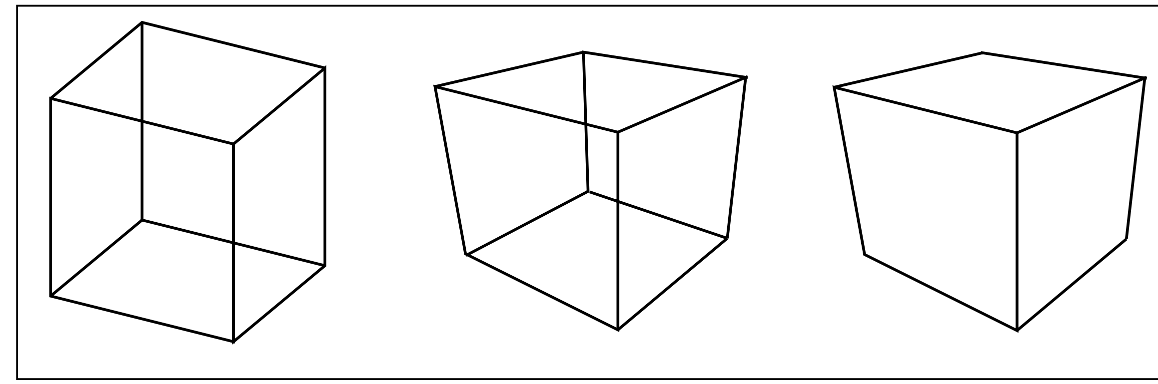
...just to show some cool graphics in between all these matrices...

$$M = M_1 M_2 ... M_n$$

$$M^{-1} = M_n^{-1} ... M_2^{-1} M_1^{-1}$$

Projections

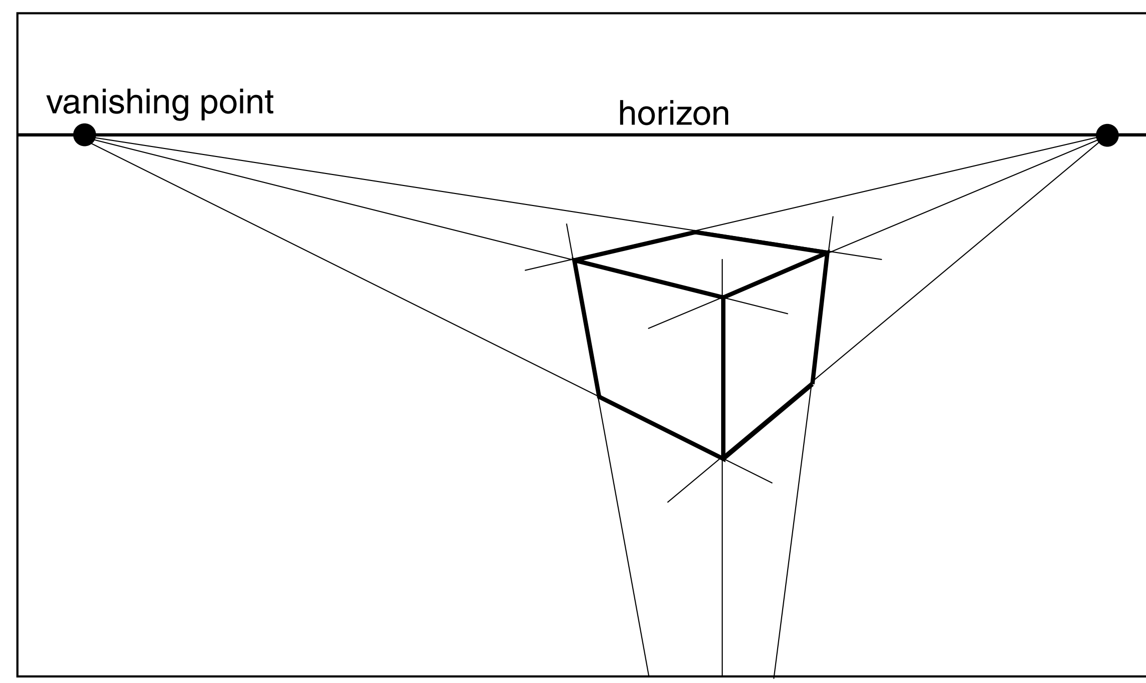
- Normally, we deal with 3D models in computer graphics
- The previous techniques explained how these models are transformed in 3D space
- At some point, the transformed 3D models have to be projected onto 2D screen space (i.e., we make pixels out of vertices and triangles)
- The following techniques explain how this 3D-to-2D transformation can be done
- It is referred to as projection
- Different types of projections exist



Orthographic projection

Perspective projection

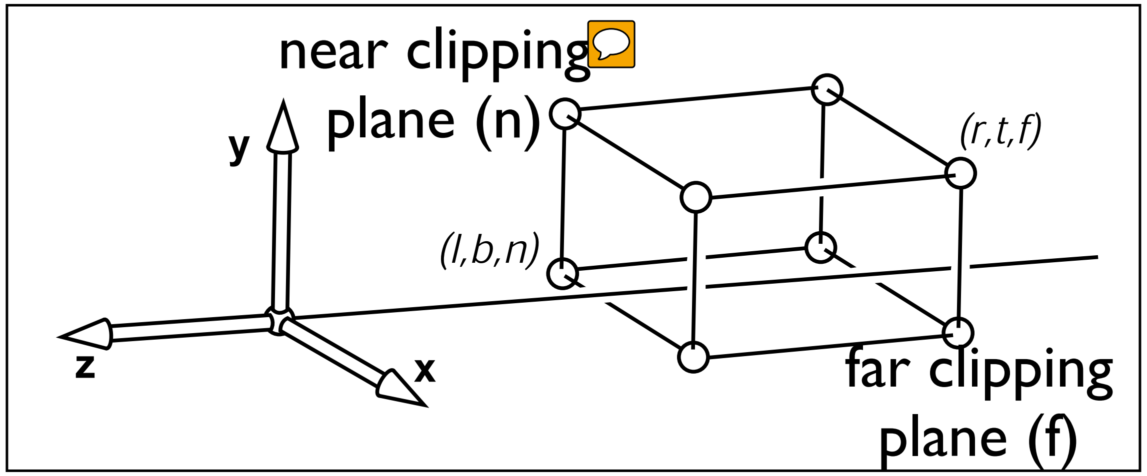
Perspective projection without hidden lines



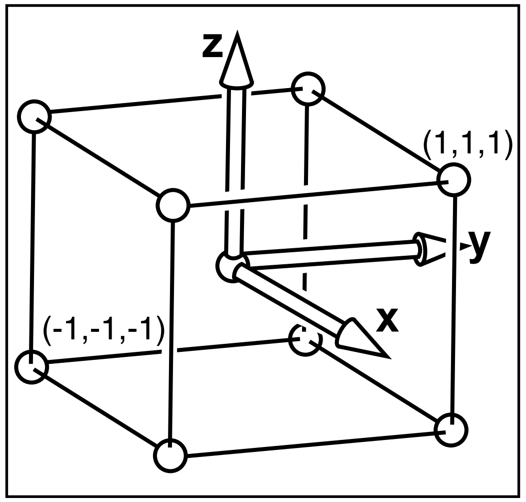
Three-point perspective with vanishing points where extended parallel lines intersect

Orthographic Projections

- Also known as parallel projection
- Is the simplest type of projection in computer graphics
- Assuming a camera/viewer is looking along the -z axis, the orthographic projection first transforms scene vertices from a user-defined scene volume (called orthographic view volume) into a normalized view volume (called canonical view volume)
- Then the z component does contain an important value, which is ignored for now, and the x and y components are transformed into screen/pixel space (0,0 - n_x,n_y)



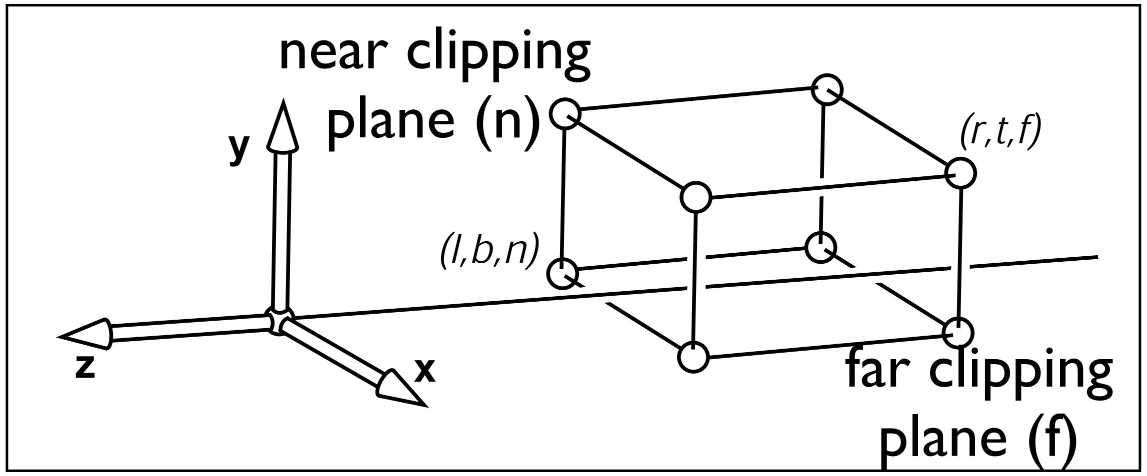
Orthographic viewing volume with defined boundaries (l,b,n and r,t,f)



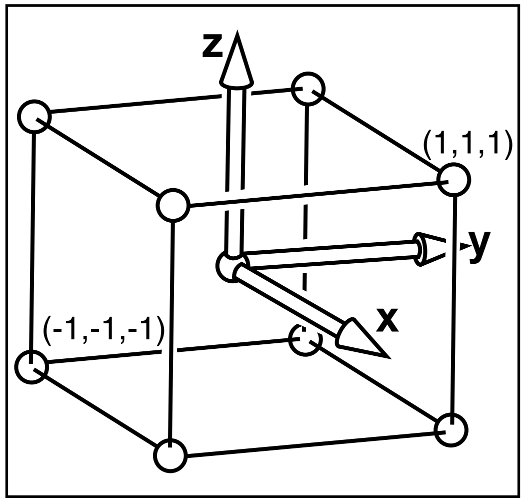
Canonical viewing volume with given boundaries (-1,-1,-1 and 1,1,1)

Orthographic Projections

- Also known as parallel projection
- Is the simplest type of projection in computer graphics
- Assuming a camera/viewer is looking along the -z axis, the orthographic projection first transforms scene vertices from a user-defined scene volume (called orthographic view volume) into a normalized view volume (called canonical view volume)
- Then the z component does contain an important value, which is ignored for now, and the x and y components are transformed into screen/pixel space (0,0 - n_x,n_y)



Orthographic viewing volume with defined boundaries (l,b,n and r,t,f)



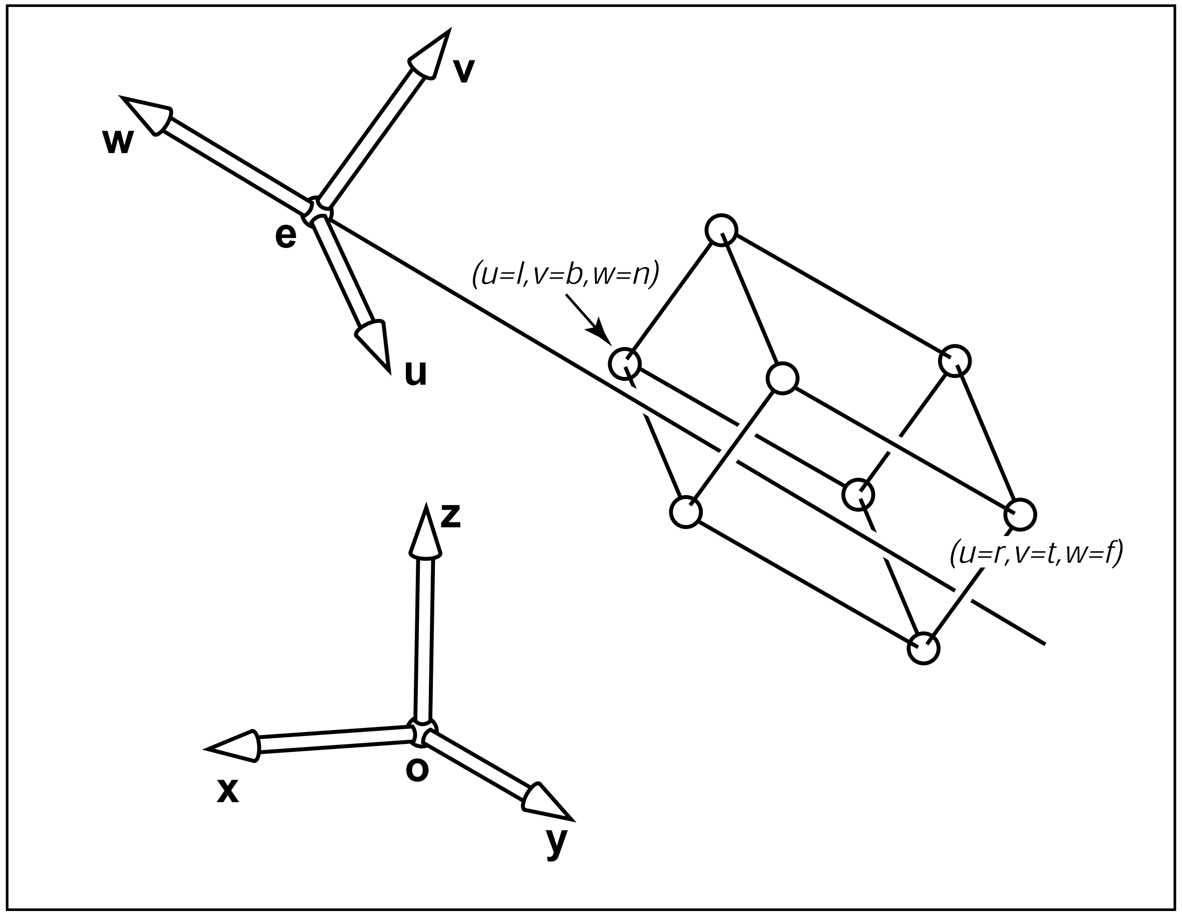
Canonical viewing volume with given boundaries (-1,-1,-1 and 1,1,1)

Viewport transformation; Orthographic projection matrix M_O

$$\begin{bmatrix} x_{pix} \\ y_{pix} \\ z_{can} \\ 1 \end{bmatrix} = \boxed{\begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}} \boxed{\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

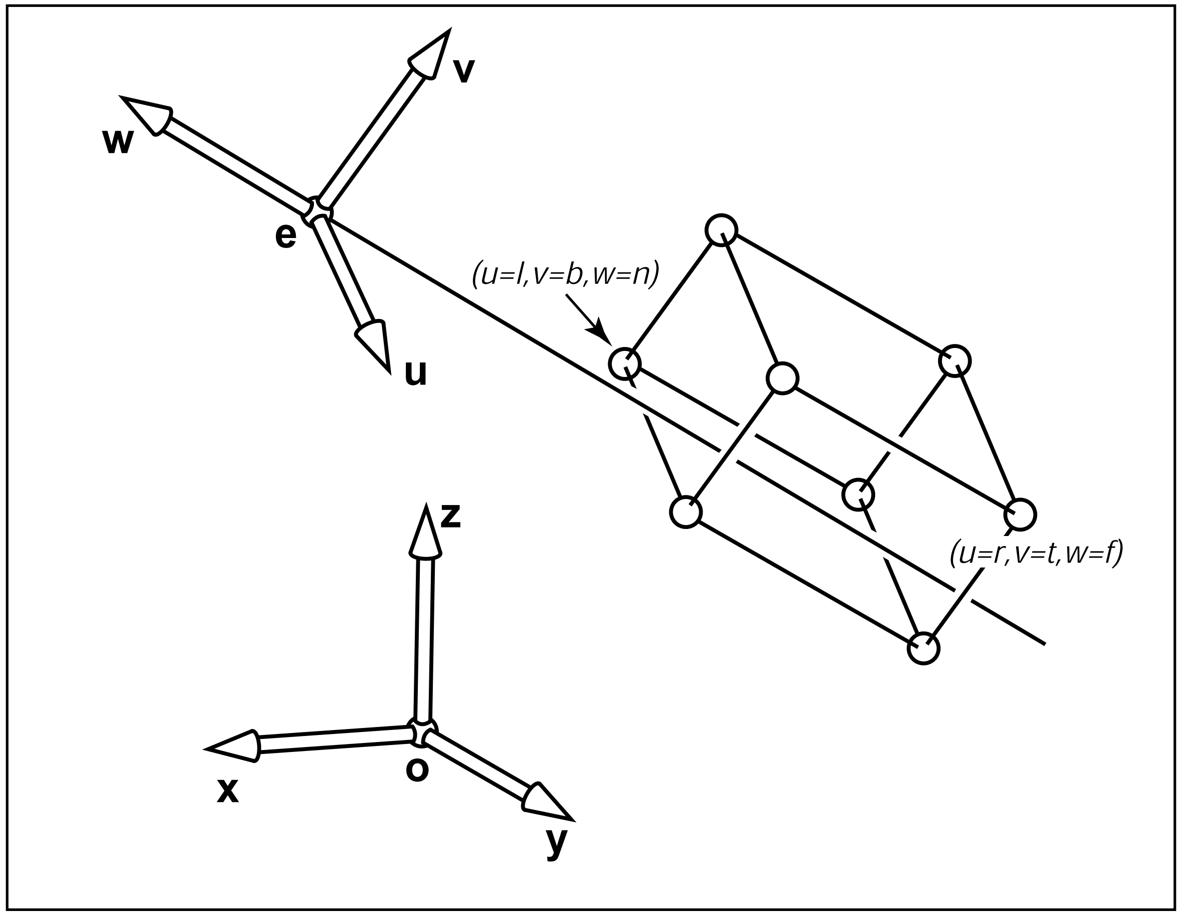
View Transformations

- What happens if the camera/ viewer is not looking along the -z axis?



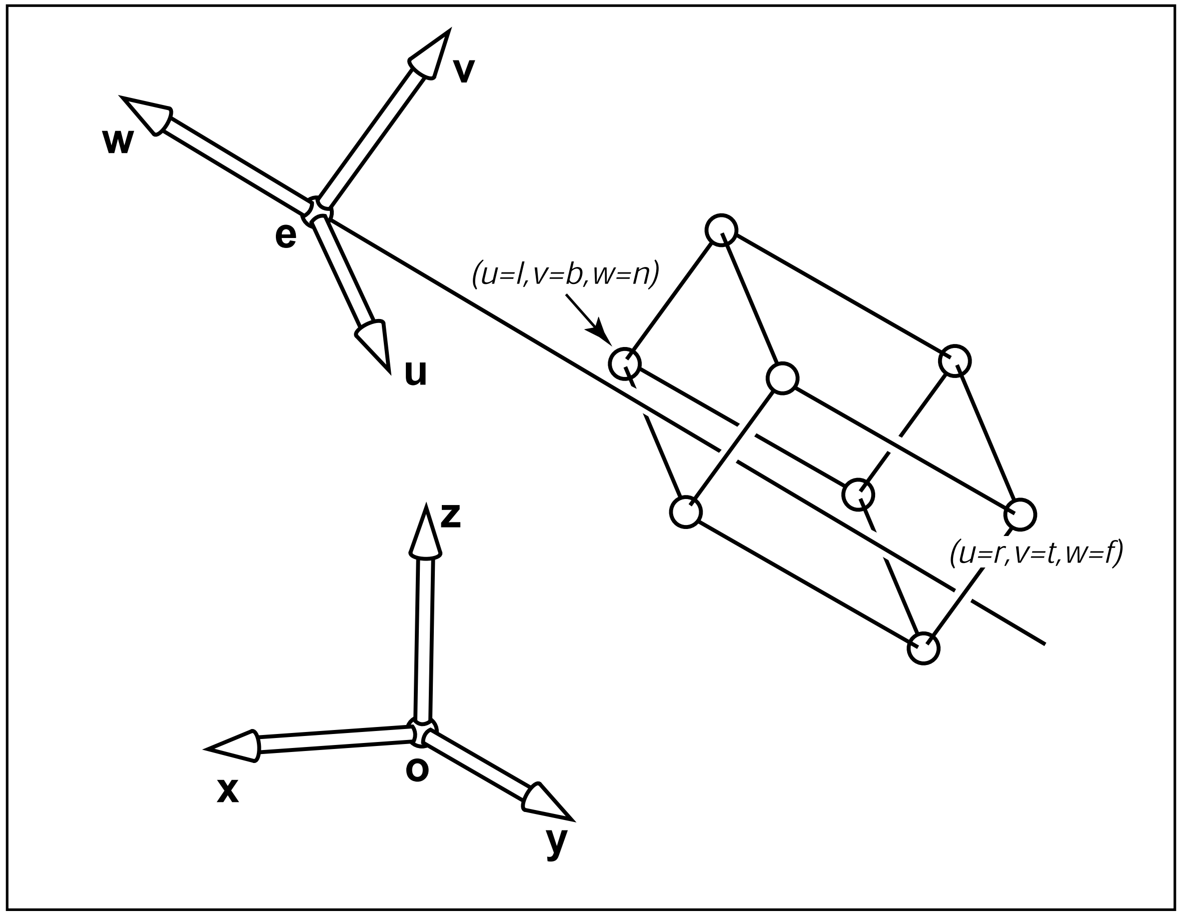
View Transformations

- What happens if the camera/ viewer is not looking along the -z axis?
- We have to apply an additional transformation to map the camera coordinate system (E) to the world coordinate system (O)



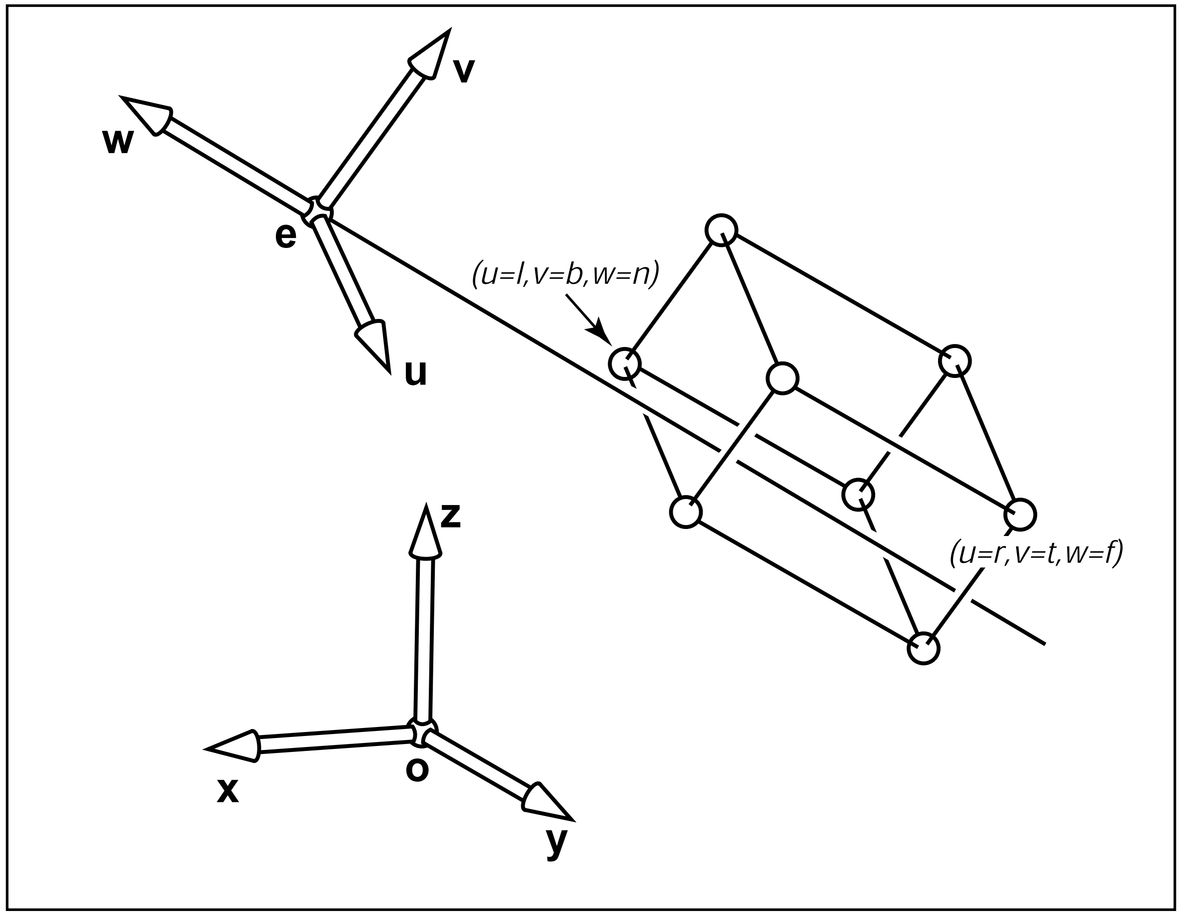
View Transformations

- What happens if the camera/viewer is not looking along the -z axis?
- We have to apply an additional transformation to map the camera coordinate system (E) to the world coordinate system (O)
- We can define our camera within O with three vectors: its position (e), its gaze direction or look-at vector (g), and its view-up direction or up-vector (t)



View Transformations

- What happens if the camera/viewer is not looking along the -z axis?
- We have to apply an additional transformation to map the camera coordinate system (E) to the world coordinate system (O)
- We can define our camera within O with three vectors: its position (e), its gaze direction or look-at vector (g), and its view-up direction or up-vector (t)
- With these parameters, we can compute the camera coordinate system E with e as origin and u,v,w as basis vectors



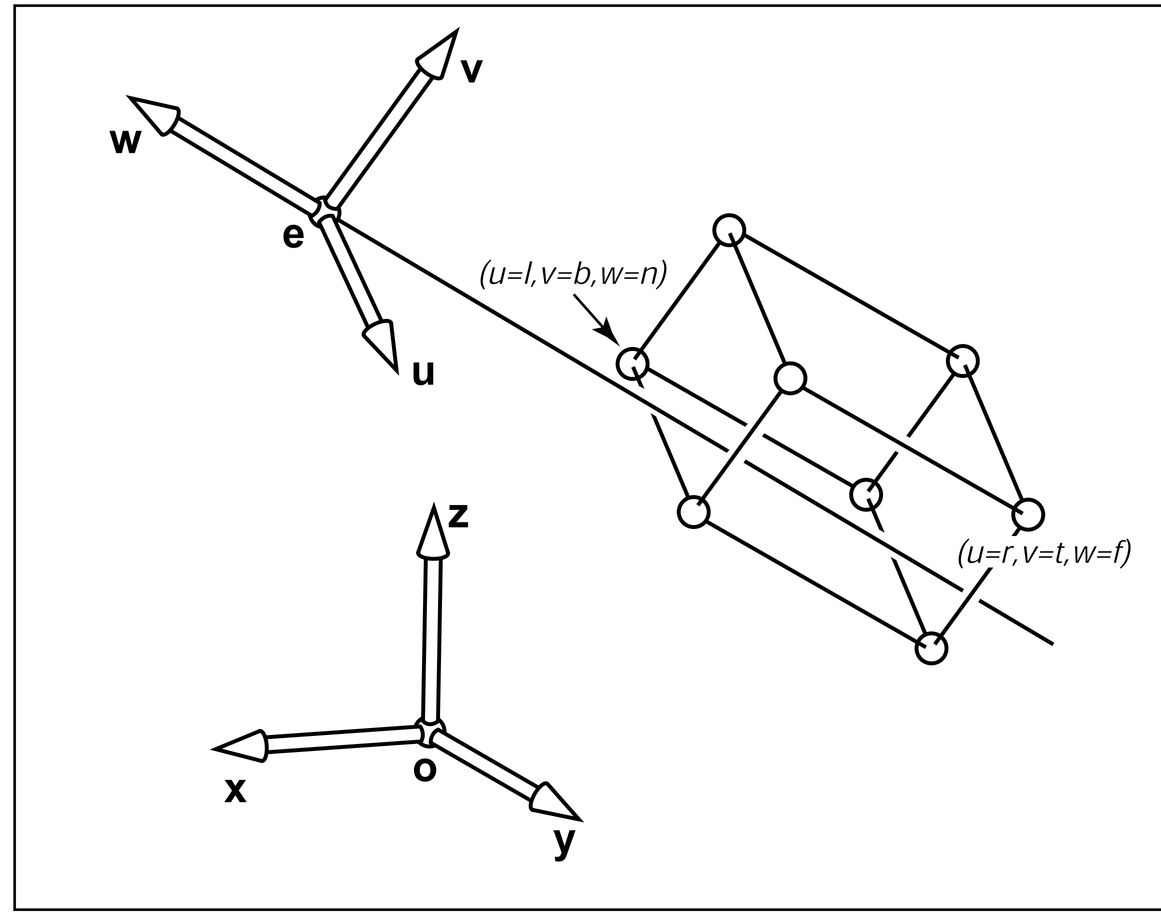
$$w = -\frac{g}{||g||}$$

$$u = \frac{t \times w}{||t \times w||}$$

$$v = w \times u$$

View Transformations

- Having the camera coordinate system, we can compute the (rigid-body) transformation that maps E to O (i.e., u,v,w to x,y,z, and e to o)
- Here, u_x indicates the x component of u, ...



$$M_v = \begin{bmatrix} u_x & u_y & u_z & -e_x \\ v_x & v_y & v_z & -e_y \\ w_x & w_y & w_z & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

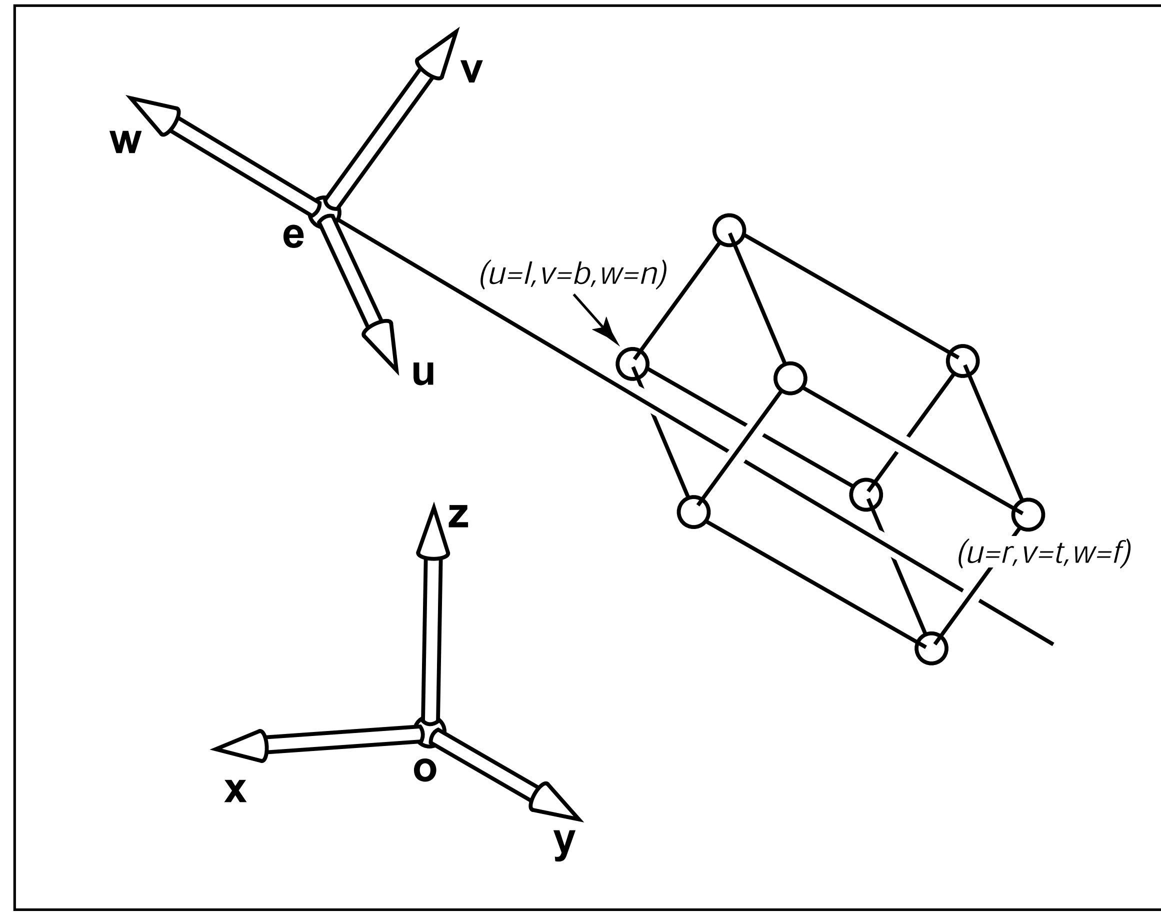
View Transformations

- Having the camera coordinate system, we can compute the (rigid-body) transformation that maps E to O (i.e., u,v,w to x,y,z, and e to o)

- Here, u_x indicates the x component of u, ...

- If we indicate the previously discussed viewport transformation with M_{vp}, and an arbitrary scene transformation with M_s, then the whole transformation process -mapping 3D object coordinates (v_{obj}) to pixels (p_s)- can be formulated as a composition (i.e., multiplication) of transformation matrices

- Later, we will call this composition "transformation pipeline"

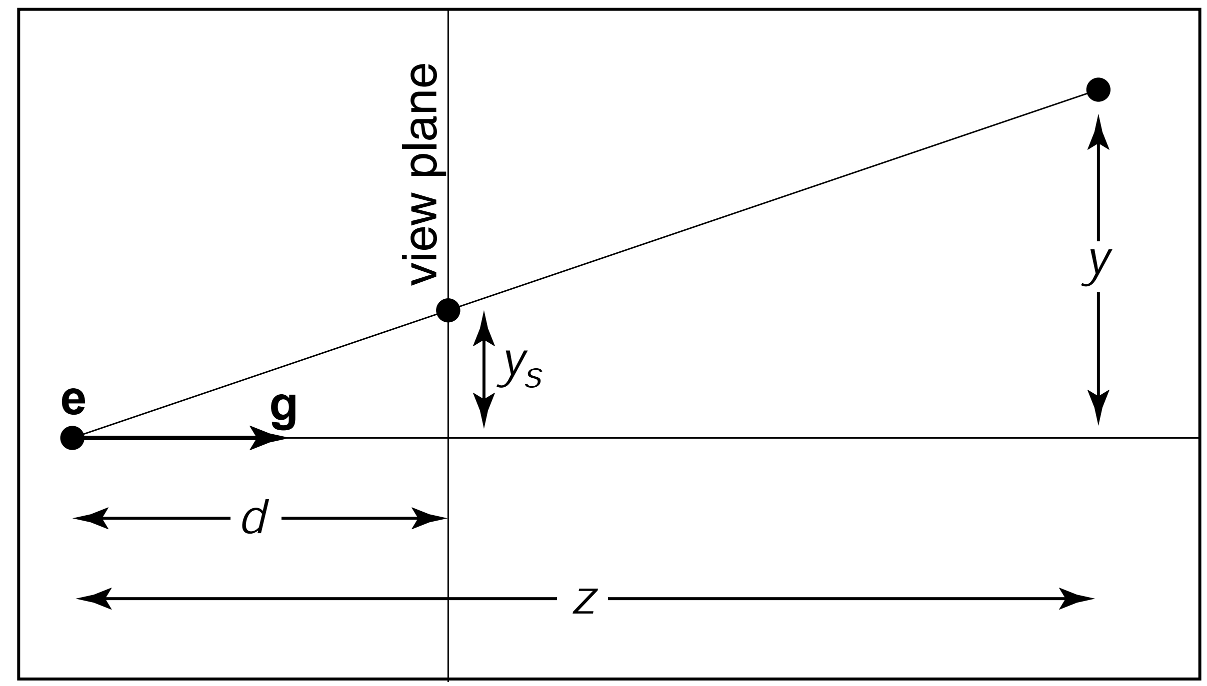


$$M_v = \begin{bmatrix} u_x & u_y & u_z & -e_x \\ v_x & v_y & v_z & -e_y \\ w_x & w_y & w_z & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$p_s = M_{vp} M_o M_v M_s v_{obj}$$

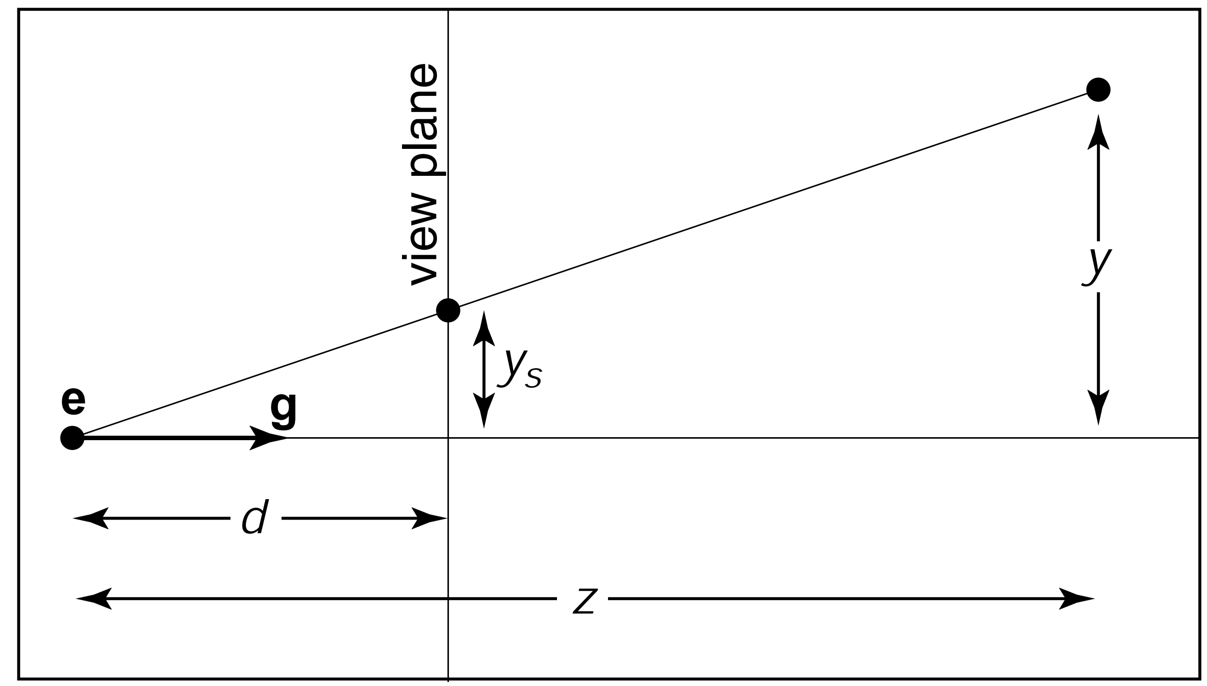
Perspective Projection

- In a perspective projection, the size of an object in the screen is proportional to it's distance
- Vertices are projected towards the camera center (e) and appear where their line of sight intersects the viewing plane at distance d (away from e)



Perspective Projection

- In a perspective projection, the size of an object in the screen is proportional to it's distance
- Vertices are projected towards the camera center (e) and appear where their line of sight intersects the viewing plane at distance d (away from e)
- Assuming again that the camera is looking along -z, this can be modeled using simple trigonometry
- How do we model this smoothly within our transformation pipeline?

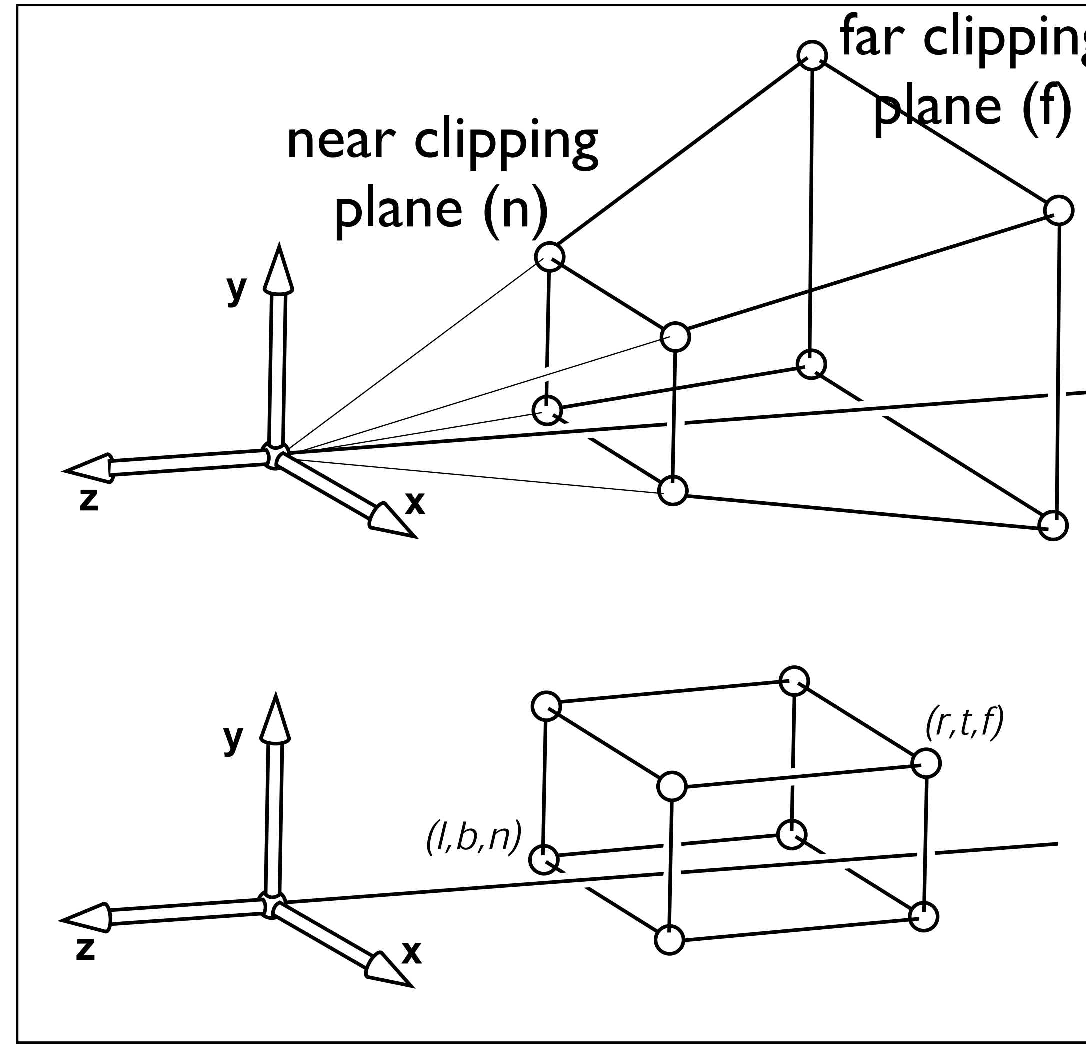


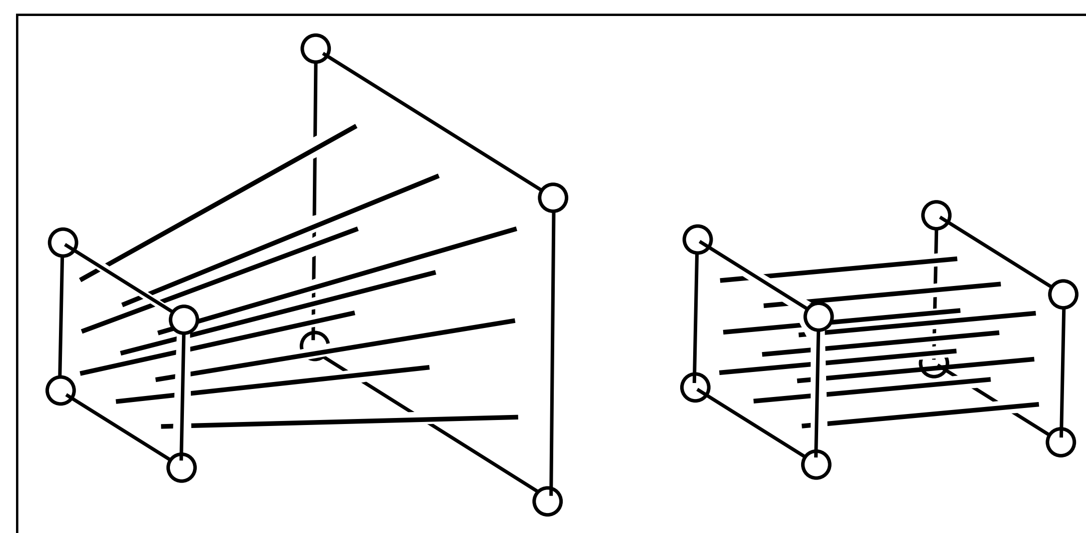
$$y_s = \frac{d}{z} y$$

$$x_s = \frac{d}{z} x$$

Perspective Projection

- Let's set the view plane to be at d=n
- A perspective transform maps any line through the origin (i.e., camera center e) to a line that is parallel to the z-axis, and without moving the point on a line at d=n
- If we apply an orthographic projection after a perspective transform, we have modeled a perspective projection
- But how do we model the 1/z part in a matrix?





Perspective Projection

- The value (f) is the distance to the far clipping plane of our viewing frustum, and (n) the distance to its near clipping plane
- Here, the homogeneous coordinate plays an important role
- If we allow the homogeneous coordinate to take up any value (not just 0 or 1), then it can be used to encode how much the other three coordinates must be scaled after a perspective transform

Perspective transform matrix M_p

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ \boxed{z} \end{bmatrix}$$

homogeneous coordinate

Perspective Projection

- The value (f) is the distance to the far clipping plane of our viewing frustum, and (n) the distance to its near clipping plane
- Here, the homogeneous coordinate plays an important role
- If we allow the homogeneous coordinate to take up any value (not just 0 or 1), then it can be used to encode how much the other three coordinates must be scaled after a perspective transform
- Dividing these three coordinates by the homogeneous coordinate is called homogenization or perspective division
- As for the orthographic case, the resulting value in the z component is not used yet - it will be used for hidden surface removal since it preserves depth order

Perspective transform matrix M_p

$$\boxed{\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ \boxed{z} \end{bmatrix}$$

homogeneous coordinate

$$\begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ z \end{bmatrix} \boxed{/z} = \begin{bmatrix} \frac{n}{z}x \\ \frac{n}{z}y \\ n+f-\frac{fn}{z} \\ 1 \end{bmatrix}$$

homogenization
or perspective division

$$M_p^{-1} = \begin{bmatrix} \frac{1}{n} & 0 & 0 & 0 \\ 0 & \frac{1}{n} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{fn} & \frac{n+f}{fn} \end{bmatrix}$$

Perspective Projection

- Finally, the perspective projection matrix (M_{pp}) can be computed as a composition of perspective transform, followed by an orthographic projection (the homogeneous coordinate (h) is not effected by M_o)

$$M_{pp} = M_o M_p$$

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

Perspective Projection

- Finally, the perspective projection matrix (M_{pp}) can be computed as a composition of perspective transform, followed by an orthographic projection (the homogeneous coordinate (h) is not effected by M_o)
- This can then be smoothly integrated into our transformation pipeline

$$M_{pp} = M_o M_p$$

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

$$p_s = M_{vp}(M_{pp} M_v M_s v_{obj})/h$$

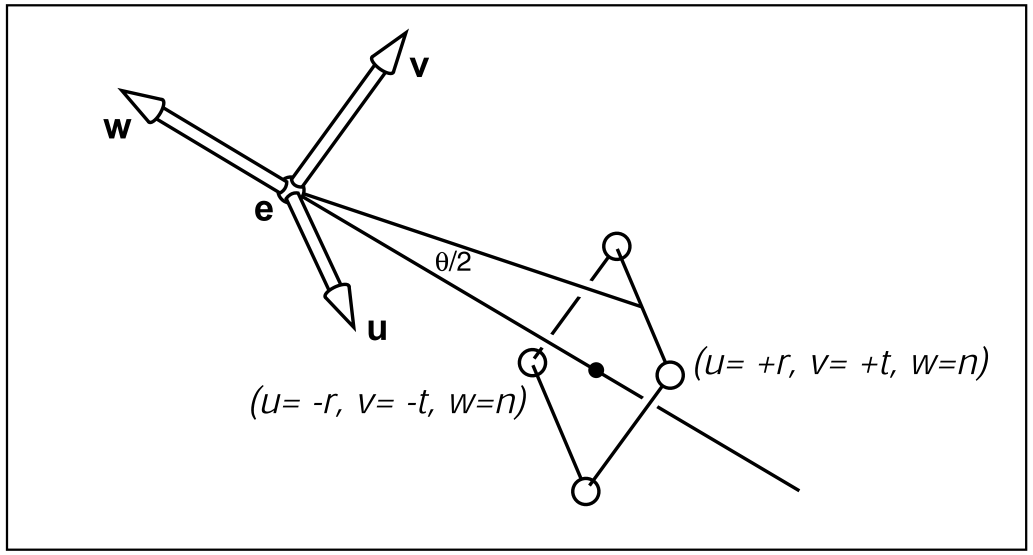
Perspective Projection

- Finally, the perspective projection matrix (M_{pp}) can be computed as a composition of perspective transform, followed by an orthographic projection (the homogeneous coordinate (h) is not effected by M_o)
- This can then be smoothly integrated into our transformation pipeline
- For an on-axis (symmetric) perspective projection, we can simplify the projection parameters and define a field of view angle while assuming square pixel footprints

$$M_{pp} = M_o M_p$$

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

$$p_s = M_{vp}(M_{pp} M_v M_s v_{obj})/h$$



$$l = -r, b = -t \qquad \frac{n_x}{n_y} = \frac{r}{t} \qquad tan\left(\frac{\theta}{2}\right) = \frac{t}{|n|}$$

NEXT ICG LAB TALK:
MARCH 15, 2016, 4:30PM

Institute of Computer Graphics

Prof. Cagatay Turkay

City University London

Interactive Visual Analysis to Aid Data-informed Analytical Problem Solving

Computer Science Building (SP3)

Room SP2 054

JKU

For more information about our talks visit **http://www.cg.jku.at/talks/invited**

Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

Thank You!