Laboratory work 9

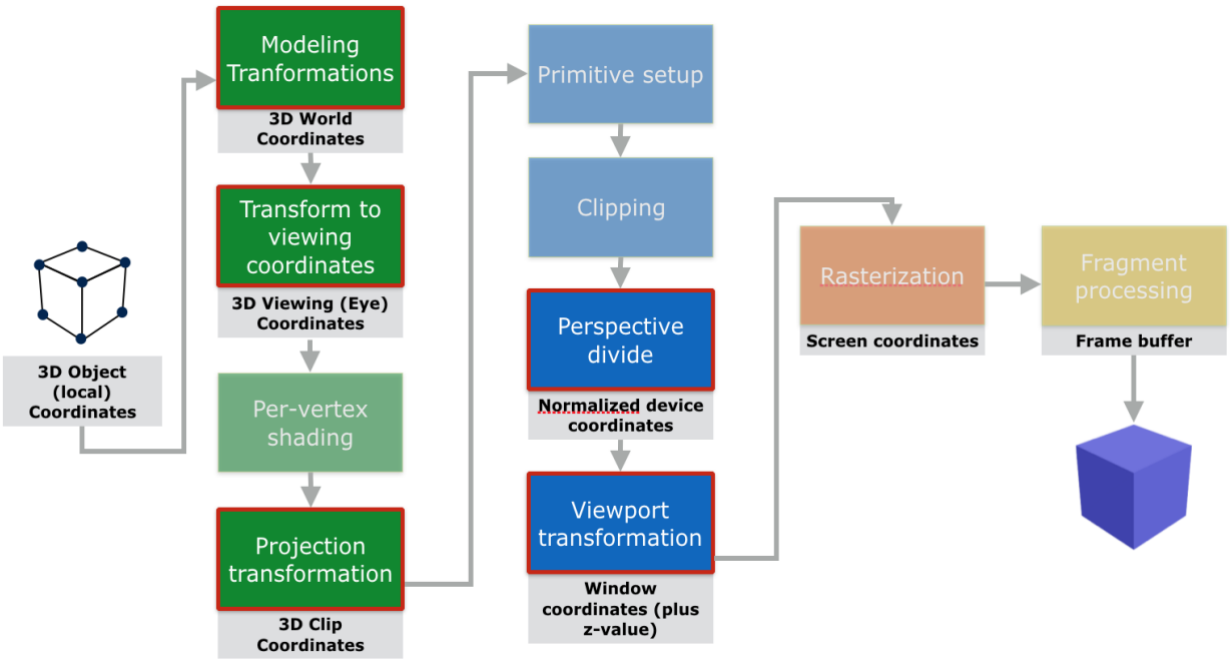
1 Objectives

This laboratory presents the topic of viewing transformations used to map 3D locations (specified by x, y, and z coordinates) to 2D coordinates (specified by pixel coordinates).

2 Visualization transformations

In order to map 3D coordinates to 2D coordinates we use a sequence of three transformations:

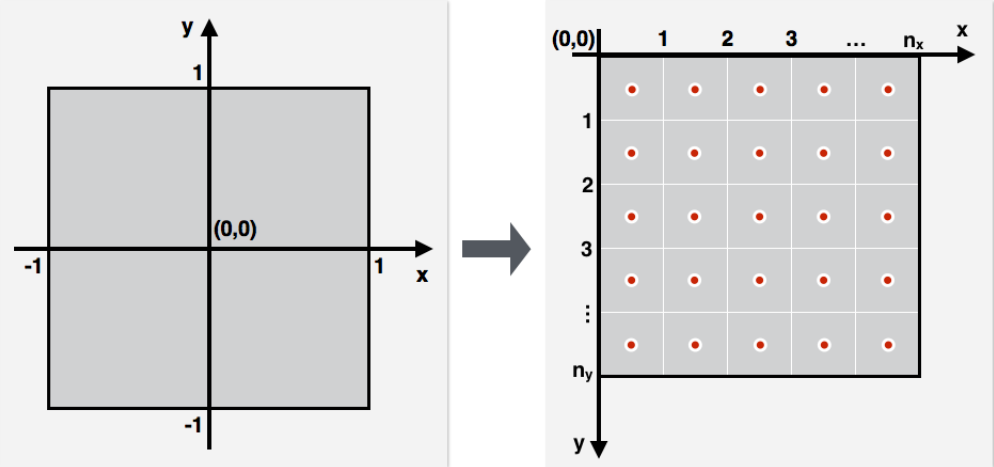
- **Camera transformation**
  - Used to place the camera at the origin and reposition all the other objects relative to the camera
  - This transformation depends only on the position and orientation of the camera
- **Projection transformation**
  - Used to project points from camera space
  - After the transformation all visible points will be in the range [−1, 1]
  - This transformation depends only on the type of projection (perspective or orthogonal)
- **Viewport transformation**
  - Used to map the unit image rectangle to the desired rectangle in pixel coordinates
  - This transformation depends only on the size and position of the output image



During these transformations we change the coordinate systems in which we specify the objects. Camera transformation changes coordinates from **world space** to **camera space**. The projection transformation moves points from **camera space** to the **canonical view volume** (here clipping is performed more efficiently). The last transformation, the viewport transformation maps the **canonical view volume** to **screen space**.

3 Viewport transformation

This transformation is used to map points from the canonical view volume (where the values are in the interval [-1, 1]) to the screen space (defined by the width and height of the resulting image). It is composed of several transformations, including translation, scale and reflection. The origin of the image is considered the top-left corner. For other specification of image space, the transformations could be different. The pixel center is considered to be at integer value plus 0.5 (both on x and y-axis).



$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & 0 \\ 0 & \frac{n_y}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

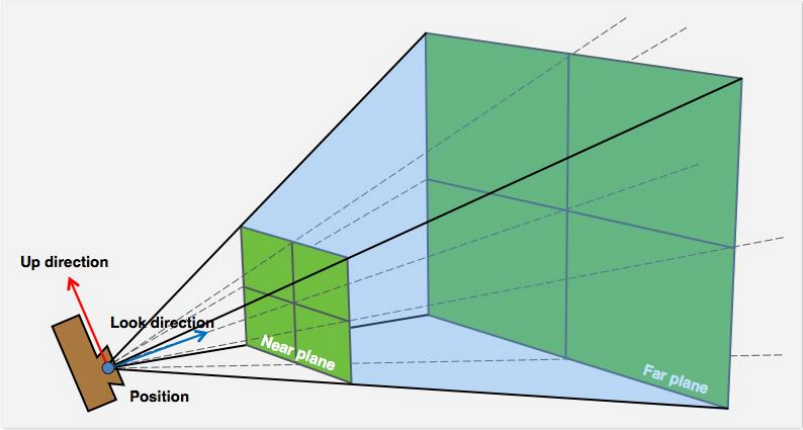
$$\mathbf{M}_{vp} = \mathbf{SMT} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x}{2} \\ 0 & -\frac{n_y}{2} & 0 & \frac{n_y}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If we want to display the image starting from another location than the origin we need to add another translation transformation to the $\mathbf{M}_{vp}$ matrix.

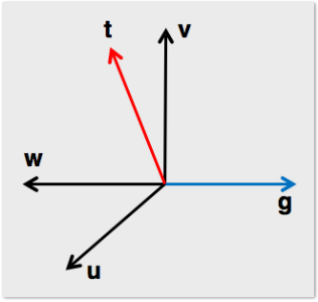
M_{vp} = T(startx, starty)*S*M*T

4 Camera transformation

A virtual camera is defined by a set of attributes and parameters such as: Camera position, Orientation, Field of view, Type of projection (perspective of parallel projection), Depth of field, Focal distance, Tilt and offset of the camera lens relative to the camera body. For a basic camera we need to specify at least the position of the camera, its orientation and the field of view.



The camera is specified in world coordinates by the following parameters: eye position **e**, gaze direction **g**, view-up vector **t**. From these parameters we need to define a coordinate system **uvw**.



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

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

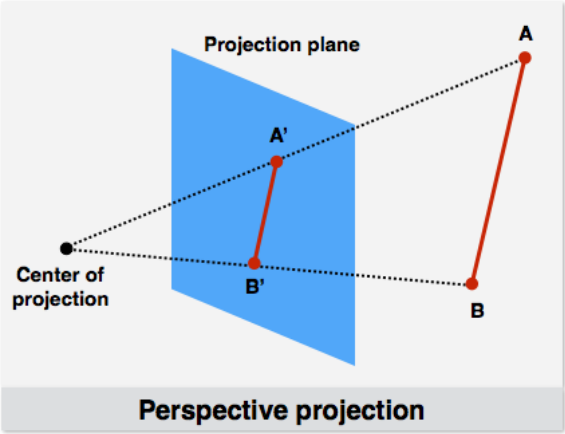
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

We need to align the two different coordinate systems (u, v, w axes with the x, y, z axes) using the following transformation matrix:

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

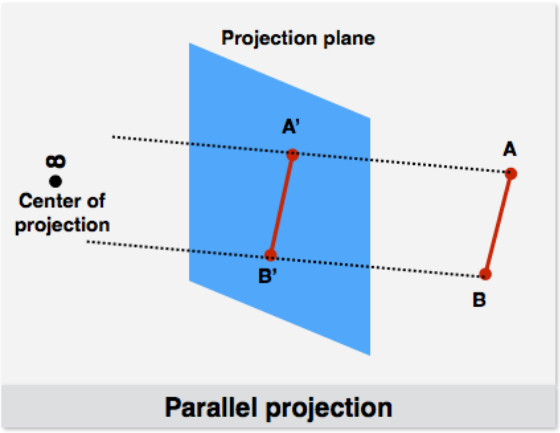
5 Projection transformation

Two types of projections are discussed in this document, perspective and parallel projections.



Perspective projection:

- Projection rays converge into the center of projection (location of the viewer)
- Objects appear smaller with the increase of distance from the center of projection (eye of observer)
- Lines parallel to the projection plane remain parallel
- Lines which are not parallel to the projection plane converge to a single point (vanishing point)



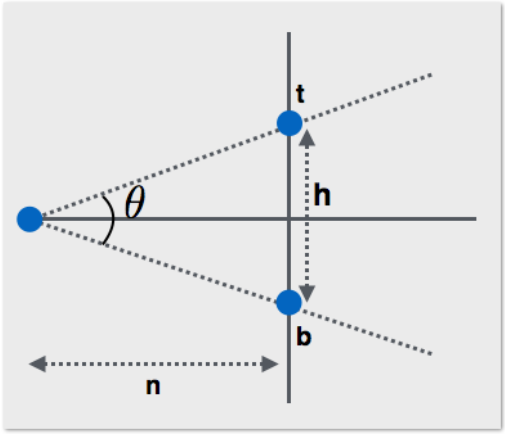
Parallel projection:

- Parallel projection rays
- Convergence point at infinity
- Viewer's position at infinity
- Parallel lines remain parallel

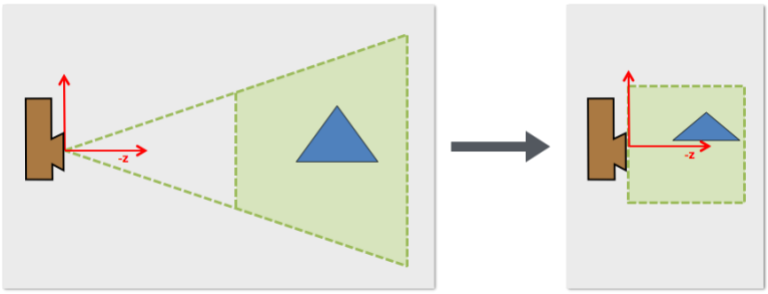
$$\mathbf{M}_{per} = \begin{bmatrix} -\frac{1}{aspect \tan\frac{\theta}{2}} & 0 & 0 & 0 \\ 0 & -\frac{1}{\tan\frac{\theta}{2}} & 0 & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\tan\frac{\theta}{2} = \frac{h}{2|n|}$$

$$aspect = \frac{width}{height} = \frac{r-l}{t-b}$$



Before displaying the vertices, we need to perform an operation called perspective divide.



The near and far planes are defined in camera coordinates. Because the camera is location in origin and is oriented in the negative z direction than the near and far values should be negative, with n > f.

$$\mathbf{P} = \mathbf{M}_p \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} xd \\ yd \\ zd \\ z \end{bmatrix} \xrightarrow{\text{After perspective divide}} \begin{bmatrix} \frac{d}{z}x \\ \frac{d}{z}y \\ d \\ 1 \end{bmatrix}$$

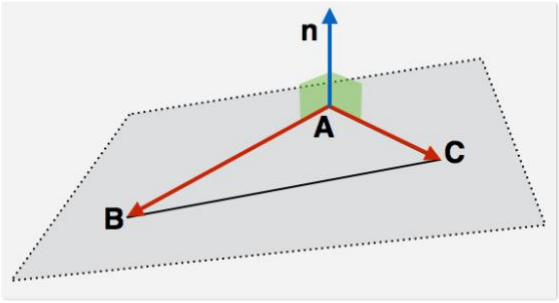
6 Combining transformations

```
construct Mvp
construct Mper
construct Mcam
M = Mvp * Mper * Mcam
for each line segment(ai,bi) do
   p = Mai
   q = Mbi
   drawline(xp/wp,yp/wp,xq/wq,yq/wq)
```

7 Compute and display the normal vector of a triangle

In order to compute the normal vector, we can use the following formula, based on the cross product between two vectors. The last step is to normalize the resulting vector in order to get a normalized vector.

$$\mathbf{n} = (B - A) \times (C - A)$$

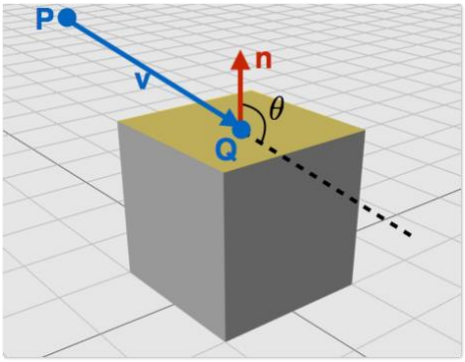


The pseudocode for displaying the normal vector:

```
displayNormalVector()
   centerPoint = compute the center point of triangle
   normalVector = compute the normal vector
   secondPoint = centerPoint + normalVector * offset
```

```
drawLine(centerPoint, secondPoint)
```

8 Back-face culling



Depending on the relative position of the camera to the object's triangles we can identify the visible triangles.

Relative position of a point **P** (camera position) against a plane (triangle):

- θ > 90˚ then P is in front of the plane
- θ = 90˚ then P is on the plane
- θ < 90˚ then P is on back of the plane

The angle can be computed using the dot product between the two vectors, **v** and **n**.

9 Basic clipping in homogeneous coordinates

We can clip the points based on the following clipping planes:

-P.w <= P.x <= P.w
-P.w <= P.y <= P.w
-P.w <= P.z <= P.w

10 Assignment

Download the source code and implement the following methods (from projection.h):

- bool clipPointInHomogeneousCoordinate(const egc::vec4 &vertex)
- bool clipTriangleInHomegeneousCoordinates(const std::vector<egc::vec4> &triangle)
- egc::vec3 findNormalVectorToTriangle(const std::vector<egc::vec4> &triangle)
- egc::vec4 findCenterPointOfTriangle(const std::vector<egc::vec4> &triangle)
- bool isTriangleVisible(const std::vector<egc::vec4> &triangle, const egc::vec3 &normalVector)
- void displayNormalVectors(egc::vec3 &normalVector, egc::vec4 &triangleCenter)
- mat4 defineViewTransformMatrix(int startX, int startY, int width, int height)
- mat4 defineCameraMatrix(Camera myCamera)
- mat4 definePerspectiveProjectionMatrix(float fov, float aspect, float zNear, float zFar)

- void perspectiveDivide(vec4 &inputVertex)