# UU Graphics

academic year 2013/14 – 4th period

## Theoretical Assignment #4:
## Rasterization – Homogeneous Coordinates & Perspective

June 03 2014

**Assignment #1:** *Homogenous coordinates – the basics, and some cool apps*

a) Consider the following matrix and vector; they operate on / represent 3D points embedded in the 4D projective space (using homogenous coordinates). Perform the matrix-vector multiplication manually to see what happens!

$$\begin{bmatrix} \lambda_1 & 0 & 0 & t_x \\ 0 & \lambda_2 & 0 & t_y \\ 0 & 0 & \lambda_3 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

b) Now for something more complicated. Find a matrix in homogenous coordinates that zooms into a 2D image (i.e., this is $\mathbb{R}^2$ and the homogenous coordinates live in $\mathbb{R}^3$) with zoom factor $\lambda \in \mathbb{R}$ and origin $\mathbf{p} \in \mathbb{R}^2$.

c) Imagine we want to implement a 2D vector drawing program, such as Powerpoint or Inkscape. We have a world coordinate system, in which have located an A4 page (with its upper left corner as the origin of the coordinate system, the positive x-axis points to the right, and the y-axis down, all units are measured in cm, and an A4 page is 21×29.7 cm in size, width × height. You also have a drawing app, running on a 1920×1080 pixel display (full-screen, landscape mode, i.e., taller side is left-right, square pixels); the origin of the coordinate system is also in the upper left corner.

   Determine a (homogenous) transformation matrix that maps the A4 page to the screen, such that the height of the paper fills the whole display height, and the virtual paper on the screen is centered around the origin.

d) Now our drawing program needs an additional feature! Implement a zoom bar – that means, compute an additional homogeneous matrix, to be multiplied with the first one, such that you can zoom in and out of the document. Important: keep the center of the virtual document fixed to the center of the screen! The factor $\lambda$ = 100% = 1.0 should yield the result of (c).

e) Then, you have to adapt your software – a new, totally cool mobile phone came out, and everybody needs to port their apps. The new phone has a 4K display (3840 × 2160, portrait mode, i.e., the taller side is up). However, due to a misunderstanding in the engineering department of the phone company, the API for the display was screwed up and it has the origin of the coordinate system in the lower left corner, with the positive x- and y-axis pointing right and upwards, respectively. What do you need to change to your solution in (c) to support the cool new phone?

f) To make the story complete – the new phone permits users to turn around the screen from portrait to landscape format. The way the user holds the phone is indicated by a flag; the screen coordinate system does not change. What needs to be adapted to get the same results in landscape mode?

**Assignment #2:** *Perspective*

a) Recap: Derive the perspective projection matrix (slides 47/48 of the lecture). Explain the main steps of the derivation.

b) We are given the following specification from the art director (the art department does not bother with orthogonal frames & stuff):

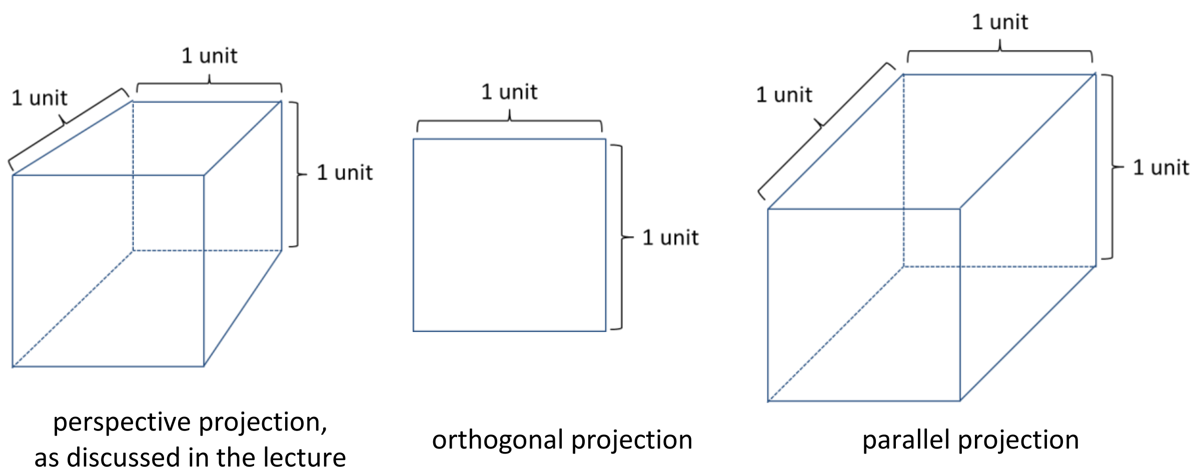
- The camera should be placed at point $\mathbf{p} = (1,2,3)$!
- It is looking at the origin! ($\mathbf{lookAt} = (0,0,0)$)
- The y-axis should be depicted as upward! ($\mathbf{up} = (0,1,0)$)

Compute an orthogonal camera coordinate frame (with the z-axis being the viewing direction)!

c) For CAD, it is often useful not to use perspective projection but orthogonal or parallel projection. Create homogenous projection matrices that do the following:

- Project the scene orthogonally in x-direction. This means, only the y- and z-axis of the world coordinates are maintained (as x- and y-axis of the screen, respectively), and the z-coordinate of the world coordinate system is just ignored (this yields a projection in z-direction without perspective, called orthogonal projection).
- Project the scene using parallel projection. Here, x- and y-axis are maintained unchanged, and the changes in the z-axis translate points diagonally. Setup a homogenous transformation matrix for this case, too.

The image below shows the different cases:




perspective projection, as discussed in the lecture

orthogonal projection

parallel projection

**Assignment #3:** *Raytracing vs. rasterization – the complexity battle, part I*

Consider the standard z-buffer algorithm and the naïve raytracing algorithm (neither of them would use any smart data structures for speedup; raytracing performs brute-force search of the randomly ordered triangles). The scene consists of *n* triangles, which cover *k* fragments (i.e., we count potentially occluded pixels again and again, if covered repeatedly, in order to determine *k*). The display overall consists of *m* pixels.

What is the asymptotic complexity of z-buffer rendering vs. raytracing, in big-O notation, dependent on these parameters?