UU Graphics

academic year 2013/14 – 4th period

Additional Practice Assignments
Lecture Summary / Final Exam preparation

June 24 2014

This assignment sheet is intended to serve two purposes:

- To further practice some analytical / theoretical skills.
  (In response to various feedback that we got.)
- To help you prepare for the final exam.
  (In place of a practice exam.)

Accordingly, the sheet provides a mixture of assignments. In particular, assignment 2 was based on student feedback. However, all assignments are good preparations for the exam (including the assignment 2) and could be relevant for the final exam (as well as all of the earlier tutorials, T1-T6).

Important remarks:

- This is not a practice exam! In particular:
- The level of difficulty might or might not be representative (the final exam could also be harder or much easier than the assignments shown here).
- The time needed for working through these assignments is different from the final exam.
- The exam might also cover different topics than the ones shown here; make sure to look at all topics of the lecture for your preparations.
- **The exam questions will look different.** The assignments here are formulated in a way such that you think about the topic and learn as much as possible, not for having clear and concise answers (as required for exam questions).
- Our intention at this point is to help you to further develop your skills, which will certainly be beneficial for the exam, but you cannot rely on this selection being comprehensive or fully representative for the exam.

Treat this more as an additional tutorial sheet, created with the intention of preparing you for the exam, but not as a replacement for studying all of the past lectures and tutorials. Nonetheless, it should be quite useful to work on these assignments.

Additional Tutorial session: There will be again three parallel tutorial session on Thursdays (June 26, after the lecture, as always). The tutors there will help you with these assignments, and can also answer questions concerning the previous topics. Do not miss this opportunity for additional personal coaching and feedback!

Good luck with your final preparations!

Assignment #1: *Homogeneous coordinates*

Consider 3D vectors represented as 4D homogeneous coordinates:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3, \text{ represented as } \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \in \mathbb{R}^4.$$

(a) The homogeneous representation distinguishes between Euclidean points and their homogeneous coordinates (representations). Please explain (in one sentence or formula), which homogeneous coordinates are representing the same Euclidean point.

Solution: Two homogeneous vectors represent the same Euclidean points if and only if they are non-zero multiples of each other.

Formally, we define a new equivalence relation ("overloaded operator=()") as follows:

$$(\mathbf{x} \equiv \mathbf{y}) \underset{\text{def.}}{\Leftrightarrow} (\exists \lambda \neq \mathbf{0} : \mathbf{x} = \lambda \mathbf{y})$$

(b) Which Euclidean points are represented by the following homogeneous coordinates?

$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 3 \\ 6 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 6 \\ 4 \\ 2 \\ 2 \end{pmatrix}$	$\begin{pmatrix} 4 \\ 0 \\ 4 \\ 4 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1.5 \\ 3 \\ 0.5 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 2 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} -1 \\ 0 \\ -1 \\ -1 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 4 \\ 6 \\ 2 \end{pmatrix}$
(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)

Solution:

$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 3 \\ 6 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 2 \\ 3 \\ 6 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$
(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)

Equivalent points are marked with the same color.

(d) A rather theoretical assignment, but not difficult at all: As introduce in the lecture, a *projective map* is a linear map in homogenous coordinates (a 4×4 matrix in our case here). Proof the following: any non-zero multiple of such a matrix (i.e., we multiply all entries simultaneously by the same non-zero number) has the same effect on the result if it is interpreted as Euclidean point.

Hint: show that

$$\forall \lambda \neq 0: \quad \lambda \cdot \mathbf{M} \cdot \mathbf{x} \equiv \mathbf{M} \cdot \mathbf{x},$$

where $\lambda \cdot \mathbf{M}$ denotes the matrix where all entries of $\mathbf{M}$ have been multiplied by λ and $\equiv$ denotes equality with respect to the Euclidean point being represented.

Solution:

The solution is actually very simple. If we scale the whole matrix, it just scales the output, which does not affect the point. Formally, we can write:

$$(\lambda \cdot \mathbf{M}) \cdot \mathbf{x} = \lambda \cdot (\mathbf{M} \cdot \mathbf{x}) \equiv \mathbf{M} \cdot \mathbf{x}$$

Assignment #2: *Homogeneous transformations [feedback]*

We now consider homogeneous transformations (projective maps) of 2D Euclidean points (3D homogeneous coordinates). Our goal is to derive various transformation matrices.

(a) Provide a homogeneous 3×3 transformation matrix that translates a 2D point $\mathbf{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ (specified in homogeneous coordinates) by a vector $\mathbf{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$.

Solution:

$$\mathbf{T}_t = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

What happens if you input a general point $\mathbf{x}' = \begin{pmatrix} wx \\ wy \\ w \end{pmatrix}$ to the same mapping – is the result still consistent?

Solution:

$$\mathbf{T}_t \begin{pmatrix} wx \\ wy \\ w \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} wx \\ wy \\ w \end{pmatrix} = \begin{pmatrix} wx + wt_x \\ wy + wt_y \\ w \end{pmatrix} = \begin{pmatrix} w(x + t_x) \\ w(y + t_y) \\ w \end{pmatrix} \equiv_{\text{div by } w} \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

So – yes it works! The scaled point is also translated by the same vector when interpreted as a Euclidean point.

(b) Write down a homogenous 3×3 transformation matrix that represents a rotation around the origin by angle α.

Solution:

$$\mathbf{R}_\alpha = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

No big change, just add a dummy w-row/-column to pass on the homogeneous parameter unchanged. Btw – any global scaling $\lambda \mathbf{R}_\alpha$ for $\lambda \neq 0$ would have the same effect, as we saw before!

(c) Now, by combining the two results, describe how we can compute a single homogeneous 3×3 transformation matrix that rotates points around an arbitrary rotation center $\mathbf{p}$ in the plane.

Solution:

Easy enough – here we go:

$$\begin{pmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & p_x\cos\alpha + p_y\sin\alpha - p_x \\ \sin\alpha & \cos\alpha & -p_x\sin\alpha + p_y\cos\alpha - p_y \\ 0 & 0 & 1 \end{pmatrix}$$

(d) Alright – let's move on to something more difficult. Our task is to create a homogenous transformation matrix that meets the following conditions:

- Again, the 3×3 matrix should operate on 2D points (when interpreted as Euclidean points), represented in 3D homogenous coordinates.
- We fix the following conditions:
  - The origin, $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, should be mapped to a 2D point $\mathbf{o}' = \begin{pmatrix} p_x \\ p_y \end{pmatrix}$ (i.e., $\begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix}$ in homogenous coordinates).
  - The point $\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$, should be mapped to a 2D point $\mathbf{u}' = \begin{pmatrix} u_x \\ u_y \end{pmatrix}$ (i.e., $\begin{pmatrix} u_x \\ u_y \\ 1 \end{pmatrix}$ in homogenous coordinates).
  - The point $\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$, should be mapped to a 2D point $\mathbf{v}' = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$ (i.e., $\begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix}$ in homogenous coordinates).

These conditions (mapping the origin and the coordinate axes) fix a (unique) affine map. Compute this affine map and represent it in homogeneous coordinates.

Solution:

The linear part of this mapping should map the x-axis to $\mathbf{u}'$ and the y-axis to $\mathbf{v}'$. This means, we get a linear map of

$$\begin{pmatrix} u_x - p_x & v_x - p_x \\ u_y - p_y & v_y - p_y \end{pmatrix}$$

in ordinary Euclidean coordinates.

In addition, the origin also needs to be shifted; we can put this in the last column of the homogeneous transformation. Overall, we get:

$$\begin{pmatrix} u_x - p_x & v_x - p_x & p_x \\ u_y - p_y & v_y - p_y & p_y \\ 0 & 0 & 1 \end{pmatrix}$$

(e) Now, one more step: We do not map the coordinate system, but we allow a general reference frame to be mapped. This is the same that we did in the lecture for deriving the mapping for texture mapping; now we want to express this in homogeneous coordinates.

Again, we want to meet the following conditions

- Once more, our 3×3 matrix should operate on 2D points (when interpreted as Euclidean points), represented in 3D homogenous coordinates.
- We fix the following additional conditions: We are given three source points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$, not co-linear (not on a single line) and map them to points $\mathbf{p}'_1, \mathbf{p}'_2, \mathbf{p}'_3$. All of these are Euclidean 2D points (3D in homogeneous coordinates).

These conditions (mapping the origin and the coordinate axes) again fix a (unique) affine map. Compute this affine map and represent the mapping in homogeneous coordinates.

Solution:

Well, we first transform back into coordinates of the first coordinate system ($\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$) and then back into the second ($\mathbf{p}'_1, \mathbf{p}'_2, \mathbf{p}'_3$) using the method developed already above in part (d). So first, retrieve the coordinates in ($\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$) by inverting the linear map and subtracting the origin:

Let $\mathbf{d}^{(1)} = \mathbf{p}_2 - \mathbf{p}_1$ and $\mathbf{d}^{(2)} = \mathbf{p}_3 - \mathbf{p}_1$ be the "coordinate axis" of the source frame, considering $\mathbf{p}_1$ as origin (any other choice would yield the same map!). Then, we form:

$$\mathbf{P}^{-1} = \begin{pmatrix} \mathbf{d}_x^{(1)} & \mathbf{d}_x^{(2)} \\ \mathbf{d}_y^{(1)} & \mathbf{d}_y^{(2)} \end{pmatrix}^{-1}$$

The full coordinate transform is given by

$$\mathbf{x} \to \mathbf{P}^{-1}(\mathbf{x} - \mathbf{p}_1) = \mathbf{P}^{-1}\mathbf{x} - \mathbf{P}^{-1}\mathbf{p}_1$$

In homogeneous coordinates, this corresponds to

$$\begin{pmatrix} \mathbf{P}^{-1} & -t_x \\ & -t_y \\ 0 \quad 0 & 1 \end{pmatrix} \text{ with } \mathbf{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix} \coloneqq \mathbf{P}^{-1} \cdot \mathbf{p}_1$$

Afterwards, we multiply with the solution of (d), getting the overall matrix of:

$$\begin{pmatrix} \mathbf{d}'^{(1)}_x & \mathbf{d}'^{(2)}_x & \mathbf{p}'_1.x \\ \mathbf{d}'^{(1)}_y & \mathbf{d}'^{(2)}_y & \mathbf{p}'_1.y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{P}^{-1} & -t_x \\ & -t_y \\ 0 \quad 0 & 1 \end{pmatrix}$$

with $\mathbf{d}'^{(1)} = \mathbf{p}'_2 - \mathbf{p}'_1$ and $\mathbf{d}'^{(2)} = \mathbf{p}'_3 - \mathbf{p}'_1$. Denoting

$$\mathbf{P}' \coloneqq \begin{pmatrix} \mathbf{d}'^{(1)}_x & \mathbf{d}'^{(2)}_x \\ \mathbf{d}'^{(1)}_y & \mathbf{d}'^{(2)}_y \end{pmatrix}, \text{ and } \mathbf{t}' = \mathbf{p}_1$$

we can also write this as

$$\begin{pmatrix} \mathbf{P}' & t_x \\ & t_y \\ 0 \quad 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{P}^{-1} & -t_x \\ & -t_y \\ 0 \quad 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{P}' \cdot \mathbf{P}^{-1} & t_y - \mathbf{P}' t_x \\ & t_x - \mathbf{P}' t_y \\ 0 \quad 0 & 1 \end{pmatrix}$$

(f) With this formula at hand, can you solve the view-port transformation problem? We are given normalized device coordinates ($[-1,1]^2$, origin in the middle, x-axis pointing to the right, y-axis pointing upward) and want to map to screen coordinates (origin upper left corner, $w \times h$ pixels, y-axis pointing downwards).

Solution:

Set $\mathbf{p}_1 = \begin{pmatrix}-1\\-1\end{pmatrix}, \mathbf{p}_2 = \begin{pmatrix}1\\-1\end{pmatrix}, \mathbf{p}_3 = \begin{pmatrix}-1\\1\end{pmatrix}$ and $\mathbf{p}'_1 = \begin{pmatrix}w/2\\h/2\end{pmatrix}, \mathbf{p}'_2 = \begin{pmatrix}w\\h/2\end{pmatrix}, \mathbf{p}'_3 = \begin{pmatrix}w/2\\0\end{pmatrix}$ and run the scheme we already have.

(g) Bonus assignment (more difficult!): The conditions in (e) fix a unique affine map, but not a unique projective map (3×3 homogeneous transformation). Why is this the case? (Why is the affine map uniquely defined, but the homogeneous one not yet?). Further, show that fixing 4 points (except from degenerate cases, such as collinear source points) fixes a projective map (when considering its action on Euclidean points; such a map is called a *homography*). Remark: the last part of this question is beyond our lecture; the "4-point" question will not show up in any exam. But it is still interesting in order to understand this better!

Solution:

An affine map is a map of the following form:

$$f(\mathbf{x}) = \mathbf{M}\mathbf{x} + \mathbf{t} = \begin{pmatrix}m_{11} & m_{21}\\ m_{21} & m_{22}\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \begin{pmatrix}t_1\\t_2\end{pmatrix}$$

It has six degrees of freedom. If we assign three points to three other points, this gives $2 \times 3 = 6$ conditions (because every point-to-point correspondence fixes two coordinates). If the source points are in general condition (not co-linear), this mapping is also sufficient; the linear map between the difference vectors $\mathbf{d}_1, \mathbf{d}_2$ (sides of the triangle spanned by the three source points) and the target vectors $\mathbf{d}'_1, \mathbf{d}'_2$ (sides of the triangle spanned by the three target points) is uniquely defined because we have prescribed the image to two basis vectors. Further, the translation is also uniquely defined by matching one more point (in our construction above, we do this explicitly for $\mathbf{p}_1 \to \mathbf{p}'_1$). Hence, we get exactly one affine map.

The homogeneous matrix has 9 entries (3×3 matrix),

$$\mathbf{M}_{hom} = \begin{pmatrix}m_{11} & m_{21} & m_{31}\\ m_{21} & m_{22} & m_{32}\\ m_{31} & m_{23} & m_{33}\end{pmatrix}$$

which yields only 8 degrees of freedom, because any global scaling of the matrix has no effect. Therefore, we can set the lower right entry always to one, without loss of generality (i.e., $m_{33} = 1$).

The remaining 8 degrees of freedom are fixed by 8 conditions that arise from assigning four 2D points to four other 2D points.

Assignment #3: *Raytracing*

Some raytracing questions: First, complexity.

(a) What is the runtime complexity of naïve raytracing (no acceleration data structures) with respect to the following to complexity parameters: n triangles, m pixels.

Answer: $\mathcal{O}(n \cdot m)$

(b) How does it compare to z-Buffering (also a naïve implementation without any culling / multi-resolution, or the similar)?

Answer: $\mathcal{O}(n + k)$, where k is the total projected area of all triangles (after clipping) measured in pixels.

(c) Can you construct a worst case scene with n triangles for which (no matter how large we prescribe n to be) the asymptotic complexity of both algorithms is the same? How realistic is this case (should we worry about it in practice)?

Answer: $k = mn$ if we use *n* triangles that fill the whole screen. In that case, the asymptotic complexity matches.

Further, geometric ray-tracing algorithms are also important!

(d) You are given a ray $\mathbf{x}(\mu) = \mathbf{p} + \mu\mathbf{t}$, that hits a surface in point $\mathbf{x}_0$ with normal $\mathbf{n}$. Compute the reflected ray. You can assume that $\mathbf{n}$ and $\mathbf{t}$ are normalized.

Solution:

Almost copying from the lecture slides, we get the direction vector of the reflected vector as

$$\mathbf{r} = 2(\mathbf{t} - \langle \mathbf{n}, \mathbf{t} \rangle \cdot \mathbf{n}) - \mathbf{t},$$

and the reflected ray equation turns into:

$$\mathbf{x}'(\mu') = \mathbf{x}_0 + \mu\mathbf{r}$$

(e) Compute the intersection of a ray and a plane: Assume you are given a ray $\mathbf{x}(\mu) = \mu\mathbf{t} + \mathbf{p}$ and a plane $\mathbf{x}(\lambda_1, \lambda_2) = \mathbf{p}_2 + \lambda_1\mathbf{t}_1 + \lambda_2\mathbf{t}_2$. Describe how you can compute the intersection point of ray and plane, or detect that no single intersection point exists. As computational primitives, you are allowed to solve linear systems (this procedure will return an error code if no solution is found).

Solution:

We want that $\mathbf{x}(\mu) = \mathbf{x}(\lambda_1, \lambda_2)$, i.e.,

$$\mu\mathbf{t} + \mathbf{p} = \mathbf{p}_2 + \lambda_1\mathbf{t}_1 + \lambda_2\mathbf{t}_2$$

which can also be written as:

$$\lambda_1\mathbf{t}_1 + \lambda_2\mathbf{t}_2 - \mu\mathbf{t} = \mathbf{p} - \mathbf{p}_2,$$

or in matrix notation as:

$$\begin{pmatrix} | & | & | \\ \mathbf{t}_1 & \mathbf{t}_2 & -\mathbf{t} \\ | & | & | \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \mu \end{pmatrix} = \mathbf{p} - \mathbf{p}_2,$$

We then have to solve this linear system. If it has no solution, the plane and the ray do not intersect in a well-defined single point. We can distinguish the case of the ray being contained in the plane with the case of the ray being parallel but offset to the plane by checking whether $\mathbf{p} - \mathbf{p}_2$ is linearly dependent on $\mathbf{t}_1, \mathbf{t}_2$. This check can be performed by checking whether

$$\det \begin{pmatrix} | & | & | \\ \mathbf{t}_1 & \mathbf{t}_2 & \mathbf{p} - \mathbf{p}_2 \\ | & | & | \end{pmatrix} =_? 0.$$

In practice, this would not be relevant; a parallel ray does not intersect the plane clearly anyways (offset or not).

Finally, put the different algorithms into perspective:

(f) Compare the shadow-mapping technique (i.e., computing a depth texture by z-Buffer rendering from the light source, then rendering with shadows in a second pass) with ray-traced shadows. What are the main advantages and disadvantages of the techniques? Write a short discussion that mentions 2-3 advantages of one technique over the other (preferably, for either one).

Possible answers (there might be more arguments to make here):

- Shadow mapping is not accurate; you can see the projections of the shadow-map texels if the resolution is too low. It is very difficult to match the resolution automatically (there is no perfect solution to this problem).
- Ray-traced shadows are usually slower than shadow maps because raytracing is usually more expensive (unless the scene is extremely large, and we use the appropriate data structures for raytracing, but no optimizations for z-buffering).
- Because shadow-map pixels are finite in extend, the problem of choosing a good offset is more difficult. For shadow rays, we just need to make sure not to collided again with the triangle where the ray started (which is much easier to check and does not involve parameter tweaking).
- Shadow maps are conceptually more involved – we need multiple rendering passes.
- Shadow maps need extra memory to store the map (in particular, this is bad if the resolution is high, and/or we have many light sources).

Assignment #4: *Perspective*

We now consider the projection matrix for perspective projection. We are given the following specification

- The screen has $w \times h$ square pixels.
- The vertical viewing angle is α.
- The camera coordinate system is the simplest one:
  - The projection center is in the origin.
  - The camera looks down the z-axis (viewing direction); the image plane is perpendicular to the z-axis.
  - The u- and v-axis of the camera coordinates are the x- and y-axis of the world coordinates (so no change here).
- In short: this is the exact same situation that we considered in the lecture for deriving the projection matrix.

(a) Write down a perspective projection matrix in homogeneous coordinates (4×4). The output for the z-coordinate can be arbitrary.

Solution:

As explained in the lecture, we get a matrix like this one (update 29.06.: highlighted):

$$\begin{pmatrix} \frac{h}{2} & 0 & 0 & \frac{w}{2} \\ 0 & -\frac{h}{2} & 0 & \frac{h}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} \frac{h/2}{\tan\left(\frac{\alpha}{2}\right)} & 0 & w/2 & 0 \\ 0 & -\frac{h/2}{\tan\left(\frac{\alpha}{2}\right)} & h/2 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

One additional remark at this point (not related to this question): The third row is arbitrary; we could choose whatever because we did not use the z-value anyways. So why do we often see matrices of this form? If we evaluate the z-coordinate (after division by w), we get:

$$z' = \frac{z-1}{z}$$

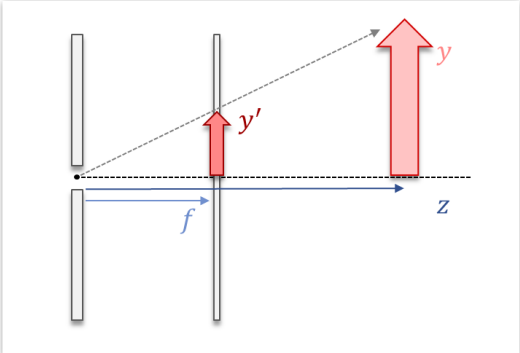
This function is strictly monotonously growing with z (it goes to 1 for large z, and to $-\infty$ for z approaching the projection plane). It is often used for z-buffering (instead of the w-coordinate, which is actually linear in depth) because it has two useful properties:

- The function grows more slowly for large values of z; so this part (further away from the viewer) is compressed. This has advantages when using low-resolution z-buffer representations (such as 16 bit fixed-point numbers, which were common on early graphics systems).
- The function has been "projected" through the division by z. It can be linearly interpolated in screen space (unlike the linear value $w' = z$, which must be interpolated as reciprocal $(w')^{-1}$, and then taking the reciprocal at each pixel again).

(b) Use the intercept theorem to explain why this matrix is correct.

Solution:

Here is the drawing from the lecture:



$$x' = f\frac{x}{z}$$

$$y' = f\frac{y}{z}$$

The intercept theorem tells us that $\frac{y'}{f} = \frac{y}{z}$, which yields the formula above. We then use this with $f = \frac{h/2}{\tan\left(\frac{\alpha}{2}\right)}$, which gives us the right scaling of the screen.

Assignment #5: *Miscellaneous*

Some additional questions concerning various topics.

(a) Shading: Describe the difference between normal interpolation ("Phong-shading") and color-interpolation ("Gouraud-shading"). Name one advantage for each method over the other.

Answer: Gouraud shading computes the lighting model (such as the Phong model) at the vertices of the triangle and interpolates the color across the triangle. Phong shading interpolates the normals of the vertices and re-executes the lighting calculation at every pixel. Before the advent of Teraflop-capable pixel-shader batteries in consumer GPUs, Phong-shading was too costly for real-time applications, because much more computation was necessary at each pixel. The model is still more costly today, but it is now feasible in real-time.

(b) Local illumination models: Write down the equation for the specular highlight in the Phong illumination model (not to be mixed up with Phong shading as name for normal interpolation, as discussed above). Describe in 1-2 sentences how the model works.

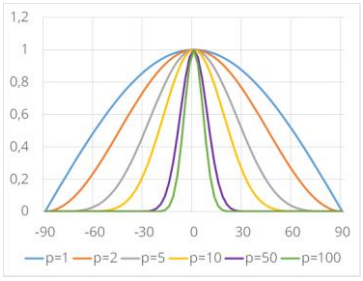
Answer: Almost copying from the slides, the specular Phong lobe is computed as

$$\langle \mathbf{r}, \mathbf{v} \rangle^p,$$

where r and v are the normalized reflection vectors and view vectors, respectively, i.e.:

$$\mathbf{r}' = 2(\langle \mathbf{n}, \mathbf{l} \rangle \cdot \mathbf{n} - \mathbf{l}) + \mathbf{l}, \qquad \mathbf{r} := \frac{\mathbf{r}'}{\|\mathbf{r}\|}$$

(and v is normalized, too). The idea is to take the cosine between perfect reflection direction and actual viewing direction and take it to a power of *p*. This power determines how large the specular highlight is – the larger p, the smaller (sharper, less glossy appearance) the highlight.



(c) You are given triangles in counter-clockwise (CCW) vertex order, i.e., when viewed from the outside, the vertices are ordered CCW. You also now the viewing direction (**w**-vector of the camera coordinates system). Describe how you can implement backface culling using difference vectors between vertices and the cross-product and dot product.

Solution:

Let $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ be the triangle vertices in CCW order. We compute an outward-pointing normal vector from it using the cross product:

$$\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)$$

We then test whether the normal is facing us or not using the dot-product:

$$\langle \mathbf{n}, \mathbf{v} \rangle > 0?$$

If the scalar product is positive, then the triangle is facing towards the camera. Otherwise, it faces backwards.

(d) You are given a 2×2 matrix with orthonormal column vectors (a so-called orthogonal matrix). Proof that the transpose of this matrix is its inverse. **Hint:** Write the product of the matrix with its transpose as multiple scalar products.

Solution:

We just perform the multiplication:

$$\begin{pmatrix} | & | \\ \mathbf{m}_1 & \mathbf{m}_2 \\ | & | \end{pmatrix} \cdot \begin{pmatrix} | & | \\ \mathbf{m}_1 & \mathbf{m}_2 \\ | & | \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} | & | \\ \mathbf{m}_1 & \mathbf{m}_2 \\ | & | \end{pmatrix} \cdot \begin{pmatrix} - & \mathbf{m}_1 & - \\ - & \mathbf{m}_2 & - \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} \langle \mathbf{m}_1, \mathbf{m}_1 \rangle & \langle \mathbf{m}_1, \mathbf{m}_2 \rangle \\ \langle \mathbf{m}_2, \mathbf{m}_1 \rangle & \langle \mathbf{m}_2, \mathbf{m}_2 \rangle \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

(all of these matrices are 2×2, the vectors $\mathbf{m}_1, \mathbf{m}_2$ denote the column vectors).

(e) Consider the simple painter's algorithm (just sorting triangles in depth order). Sketch an example scene for which this algorithm cannot work correctly, no matter how we sort the triangles (i.e., we must cut triangles into pieces in order to get the correct solution). Explain your sketch briefly.

Solution:

Here we go – these three triangles are cyclically overlapping; no sorting exists that resolves visibility: