Prof. Dr. Leif Kobbelt
Dr. Jan Möbius
Patrick Schmidt, Moritz Ibing

Basic Techniques in Computer Graphics

Assignment 3

Date Published: October 29th 2019,   Date Due: November 05th 2019

- All assignments (programming and text) have to be completed in teams of 3–4 students. Teams with fewer than 3 or more than 4 students will receive no points.
- Hand in **one solution per team per assignment**.
- Every team must work independently. Teams with identical solutions will receive no points.
- Solutions are due 14:30 on November 05th 2019. Late submissions will receive zero points. No exceptions!
- Instructions for **programming assignments**:
  - Download the solution template (a zip archive) through the Moodle course room.
  - Complete the solution.
  - Prepare a new zip archive containing your solution. It must contain exactly those files that you changed. **Only change those files you are explicitly asked to change in the task description.** The directory layout must be the same as in the archive you downloaded.
  - Upload your zip archive through Moodle before the deadline. Use the Moodle group submission feature. Only in the first week (when Moodle groups have not been created yet), list all members of your group in the file `assignmentXX/MEMBERS.txt`. Remember, only one submission per group.
  - Your solution must compile and run correctly **on our lab computers** using the exact same `Makefile` provided to you. Do not include additional libraries and do not change code outside of the specified sections. If it does not compile on our machines, you will receive no points.
- Instructions for **text assignments**:
  - Prepare your solution as a single pdf file per group. Submissions on paper will not be accepted.
  - If you write your solution by hand, write neatly! Anything we cannot decipher will receive zero points. No exceptions!
  - Add the names and student ID numbers of all team members to every pdf.
  - Unless explicitly asked otherwise, always justify your answer.
  - Be concise!
  - Submit your solution via Moodle, together with your coding submission.

Exercise 1 Meaningful Geometric Operations **[8 Points]**

For any i, let $\mathbf{p}_i$ be a **point** at location $\begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^\mathsf{T}$. Use extended coordinates to explain whether each of the following operations yields either a **point**, a **vector**, or is **not geometrically meaningful**.

(a) **[2 Points]**

$$\mathbf{p}_1 - 2\mathbf{p}_2 + \mathbf{p}_3$$

(b) **[2 Points]**

$$\mathbf{p}_0 + \sum_{i=1}^{n} (\mathbf{p}_i - \mathbf{p}_0)$$

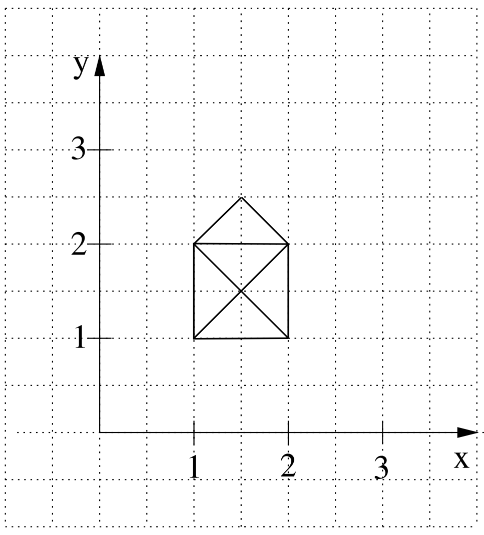
(c) **[2 Points]**

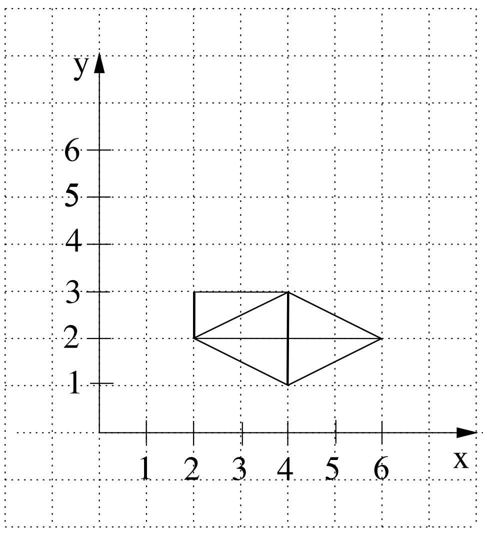
$$\alpha\mathbf{p}_0 + \beta\mathbf{p}_1, \quad \text{with } \alpha, \beta \in \mathbb{R}$$

(d) **[2 Points]**

$$\alpha\mathbf{p}_0 + \beta\mathbf{p}_1 + (1 - \alpha - \beta)\mathbf{p}_2, \quad \text{with } \alpha, \beta \in \mathbb{R}$$

Exercise 2 Linear & Affine Transformations **[10 Points]**





(a) **[8 Points]**

Derive four transformation matrices (translation, rotation, scaling and translation) that together transform the house depicted in the left image to the house depicted in the right image. Also specify the final transformation matrix. Remember to use extended coordinates! By convention, points are multiplied from the right side!

Hint: Remember that $\sin 45° = \cos 45° = \sin 135° = -\cos 135° = \frac{1}{\sqrt{2}} = \frac{\sqrt{2}}{2}$.

(b) **[2 Point]**

Specify how the standard basis and the origin are transformed by this mapping.

Exercise 3 Transformations **[6 Points]**

In this exercise we consider the transformation of one orthonormal coordinate system into another orthonormal coordinate system.

Let $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2 \in \mathbb{R}^3$ be three vectors that form an orthonormal coordinate system, i.e. $\mathbf{p}_0^\mathsf{T}\mathbf{p}_1 = \mathbf{p}_1^\mathsf{T}\mathbf{p}_2 = \mathbf{p}_2^\mathsf{T}\mathbf{p}_0 = 0$, and $\|\mathbf{p}_0\| = \|\mathbf{p}_1\| = \|\mathbf{p}_2\| = 1$. Similarly, let $\mathbf{q}_0$, $\mathbf{q}_1$, $\mathbf{q}_2 \in \mathbb{R}^3$ also be three vectors that form an orthonormal coordinate system.

(a) Inverse of an Orthonormal Matrix **[2 Points]**

Show that $\mathbf{P}^\mathsf{T} = \mathbf{P}^{-1}$, where $\mathbf{P} = \begin{pmatrix}\mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2\end{pmatrix}$, i.e. the colums of $\mathbf{P}$ are formed by the vectors $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$.

(b) Rigid Transformations **[2 Point]**

Derive the linear transformation matrix $\mathbf{M} \in \mathbb{R}^{3\times3}$ that maps $\mathbf{p}_i$ to $\mathbf{q}_i$ for all $i \in \{0, 1, 2\}$. Use the matrices $\mathbf{P}$ and $\mathbf{Q}$ from part (a).

(c) Inverse of Rigid Transformations **[2 Points]**

Show that $\mathbf{M}^\mathsf{T} = \mathbf{M}^{-1}$. Use the property shown in (a).

Exercise 4 Programming: Transformations **[16 Points]**

In this hands-on exercise, your task is to write a computer program simulating a simple 2D race track. We provide you with a function that draws a circle in a specified color. All you have to do is place differently transformed (and colored) versions of this circle into the scene.

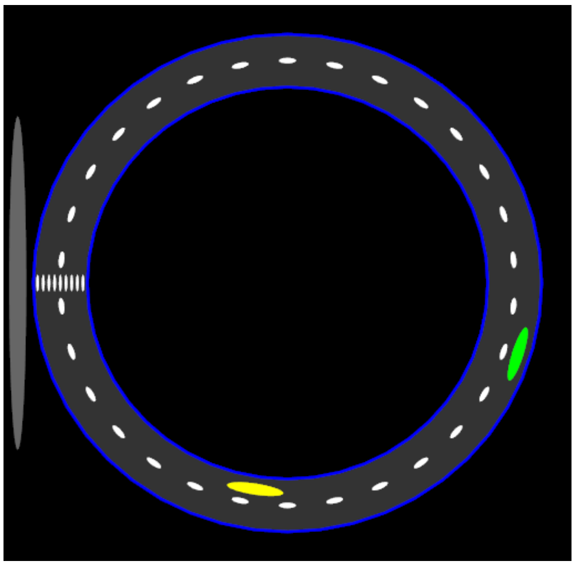
The draw routine is implemented in `assignment.cpp`. Only make modifications to that file! Do *not* modify any other file.

Each time the draw routine is called, you will have to compute the respective transformation matrices *by hand* (GLM does already offer function to create translation, rotation etc. matrices but you are *not allowed* to use them in this exercise. Instead, enter the values into the matrices yourself to show that you understood how they work. You can write your own helper functions for setting up different types of matrices. You can use the build-in matrix multiplications of GLM.). You can also use the matrix and vector classes from `glm`, e.g. `glm::mat4` or `glm::vec3` for colors.

The `drawScene()` routine gets two parameters, `scene` and `runTime`. You can neglect the first parameter as it is not used in this exercise. The second parameter holds the number of seconds passed since the first call (as floating point representation).

Notes:

- A non-transformed circle has a radius of 1.
- The local coordinate system is defined from -1 to 1.
- Transformations will be represented as 4-by-4 matrices in extended coordinates. For now, we are only interested in transformations within the x-y-plane, so the z-coordinate should stay untouched.
- **GLM stores matrices in the *column-major* format**. This means that the constructor `glm::mat4(float x0,...float x15)` uses the first four arguments `x0,...x3` to build the first *column* of the matrix! This means that you either have to write down your matrices in a transposed form or call `glm::transpose` after construction.
- When `b` is pressed, the time passes faster, undo this by pressing `a` (useful for debugging).



(a) **[4 Point]**

Draw the track itself as a blue circle in which you draw a grey circle which is a little smaller, then another blue one and a black in the center to only let the outline of the blue one be visible.

(b) **[2 Point]**

Draw a stand for the spectators on the left of the track in grey.

(c) **[4 Point]**

Draw a start / finish line as nine white ellipses.

(d) **[2 Point]**

Add a white dotted line between the two lanes as ellipses.

(e) **[4 Point]**

Add the two race cars which race around the track. The outer car should be twice as fast as the inner car. The cars race clockwise.