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

Basic Techniques in Computer Graphics

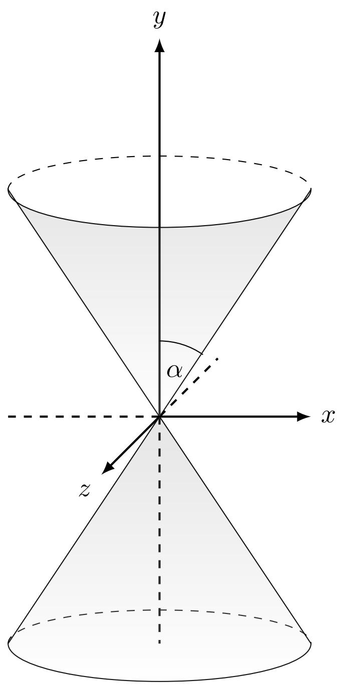
Assignment 4

Date Published: November 10th 2021, Date Due: November 17th 2021

- All assignments (programming and theory) 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 14:15 on November 17th 2021 via Moodle. Late submissions will receive zero points. No exceptions!
- Instructions for **programming assignments**:
  - Make sure you are part of a Moodle group with 3-4 members. See "Group Management" in the Moodle course room.
  - Download the solution template (a zip archive) through the Moodle course room.
  - Unzip the archive and populate the `assignmentXX/MEMBERS.txt` file. The names and student ids listed in this file **must match** your moodle group **exactly**.
  - Complete the solution.
  - Prepare a new zip archive containing your solution. It must contain exactly the files that you changed. **Only change the files you are explicitly asked to change in the task description.** The directory layout must be the same as in the archive you downloaded. (At the very least it must contain the `assignmentXX/MEMBERS.txt`.)
  - One team member uploads the zip archive through Moodle before the deadline, using the group submission feature.
  - Your solution must compile and run correctly **on our lab computers** by only inserting your **assignment.cc** and **shader files** into the Project. If it does not compile on our machines, you will receive no points. If in doubt you can test compilation in the virtual machine provided on our website.
- 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 Explicit and Implicit Representations of Surfaces [15 Points]

Consider the following double cone with (half) opening angle α. The cone is centered at the origin and extends infinitely in y and $-y$ direction.



(a) Explicit Representation [3 Point]

Derive an explicit (a.k.a. parametric) representation of the double cone depicted above, i.e. a function $f_c(...) \in \mathbb{R}^3$ enumerating all points of the surface. Don't forget to specify the parameters of f_c (including their range). Explain your derivation!

(b) Implicit Representation [2 Point]

Derive an implicit representation of the double cone depicted above, i.e. a function $F_c(x, y, z) \in \mathbb{R}$ such that $F_c(x, y, z) = 0$ if and only if the point $(x, y, z)^\mathsf{T}$ lies on the surface. Explain your derivation!

(c) Quadrics [2 Point]

Express your implicit representation from part (b) as a quadric. I.e. specify the matrix $\mathbf{Q}_c \in \mathbb{R}^{4\times4}$ such that $(x, y, z, 1) \cdot \mathbf{Q}_c \cdot (x, y, z, 1)^\mathsf{T} = F_c(x, y, z)$.

(d) Normals of Implicit Surfaces [4 Point]

Consider a general quadric

$$\mathbf{Q} = \begin{pmatrix} 2a & b & c & d \\ b & 2e & f & g \\ c & f & 2h & i \\ d & g & i & 2j \end{pmatrix}$$

which defines a surface via $(x, y, z, 1) \cdot \mathbf{Q} \cdot (x, y, z, 1)^\mathsf{T} = F(x, y, z) = 0$.

One often needs to determine surface normal vectors (orthogonal to the surface), e.g. to perform lighting computations. This can be done by computing the gradient $\nabla F = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right)^\mathsf{T} \in \mathbb{R}^3$ of the implicit function: At a given point the gradient is always orthogonal to the level set (surface) of the function.
The surface normal at point $(x, y, z)^\mathsf{T}$ can then be defined by normalizing the gradient:

$$\mathbf{n}(x, y, z) = \frac{\nabla F(x, y, z)}{||\nabla F(x, y, z)||} \in \mathbb{R}^3.$$

Derive the entries of matrix $\mathbf{G} \in \mathbb{R}^{3\times 4}$ which computes the gradient at a given point (x, y, z), i.e. such that $\mathbf{G} \cdot (x, y, z, 1)^\mathsf{T} = \nabla F(x, y, z)$.
Hint: Start by writing down the partial derivatives of F.

(e) Example **[2 Points]**

An implicit surface is specified by

$$F_e(x, y, z) = x(x - 2) + y(y - 4) + 3z(2\sqrt{3} - z) - 4 = 0$$

Specify the (unique) symmetric quadric $\mathbf{Q}_e$ such that $(x, y, z, 1) \cdot \mathbf{Q}_e \cdot (x, y, z, 1)^\mathsf{T} = F_e(x, y, z)$.

(f) Example **[2 Points]**

Using your expression for matrix $\mathbf{G}$, compute the normal at point $\mathbf{p} = (1, 5, 2\sqrt{3}, 1)^T$ of the surface defined by $\mathbf{Q}_e$.

Exercise 2 Frustum Transformation for Orthogonal Projections **[7 Points]**

In contrast to perspective projections, in orthogonal projections all viewing rays are orthogonal to the image plane. As a result, the viewing frustum of an orthogonal projection has a different shape as the one of a perspective projection. Also, the frustum transformation is much simpler for orthogonal projections. In this exercise you will derive both, the shape of the viewing frustum and the matrix of the frustum transformation.

(a) Shape of the Frustum **[1 Point]**

What geometric shape does the viewing frustum of an orthogonal projection have if both, the near and the far plane are parallel to the image plane? Explain your reasoning in a few sentences and name the resulting shape. No formulas needed here.

(b) Transformation Matrix **[6 Points, 2 per matrix]**

Similarly to a perspective projection, the viewing frustum of an orthogonal projection can be uniquely defined by a far plane, a near plane and a rectangle on the near plane. Let the near and far planes be orthogonal to the z-axis at $z_{\text{near}} = -n$ and $z_{\text{far}} = -f$. Let the rectangle on the near plane be given by the top coordinate t, the bottom coordinate b, the left coordinate l and the right coordinate r. Derive the frustum matrix $\mathbf{M} \in \mathbb{R}^{4\times 4}$ that maps this frustum to the cube $[-1, 1]^3$ as a combination of a translation $\mathbf{T}$ and a scaling $\mathbf{S}$. Specify the matrices of both transformations as well as the final transformation matrix. Remember that the near plane is mapped to the plane orthogonal to the z-axis at $z = -1$.

Exercise 3 Programming **[18 Points]**

In this task you are going to add a third dimension to last week's 2D racing "game". We already went ahead and replaced the circles by spheres and implemented the game logic and rendering, including simple lighting, for you.

All functions you need to change are implemented in `assignment.cpp`. Only make modifications to that file! Do *not* modify any other file in the folder.

(a) Look-at Transformation **[4 Points]**

Complete the lookAt (...) function. It gets camera position, viewing direction, and up-vector as parameters and should set up and return a view matrix of type glm::mat4 that performs the corresponding lookAt transformation (see lecture) – such that afterwards everything is in a standard projection setting. Remember that you have to normalize the forward, up and right vector before constructing the lookAt matrix to avoid scaling effects.

Note: You have to set the matrix entries yourself. *Do not* use any of the convenience functions that do the work for you! You may define multiple matrices and use the predefined matrix multiplication operator to combine them.

(b) Frustum Transformation **[4 Points]**

Complete the buildFrustum (...) function. Its arguments are the vertical field of view angle ϕ in degrees (phiInDegree), the screen's aspect ratio aspectRatio, and the near and far plane distances near and far. Complete this function so it sets up and returns a matrix that performs the frustum transformation according to these parameters (see lecture slides).

Note: Just as in Exercise (a), set the matrix entries yourself!

(c) Camera Transformation 1 **[2 Points]**

The race track revolves around the origin. To get a good look at it, call the lookAt (...) function near the beginning of drawScene (...) (the comments in the source code indicate where exactly) to create a view matrix that positions the camera at $\begin{pmatrix}0 & -1 & 1\end{pmatrix}$, looking towards the origin. Store this matrix in the global variable viewMatrix (which is then used for the track rendering in the routines we already implemented for you).

(d) Using the Frustum Transformation **[4 Points]**

The buildFrustum (...) function needs to be called and the returned matrix has to be stored to the global variable projectionMatrix. This needs to be done not only once, but every time the user resizes the window because that changes the dimensions and the aspect ratio of the frustum. In our framework the function resizeCallback (...) gets called whenever the window gets resized (and also once at startup). Hence, this is the perfect place for you to set projectionMatrix. Use a vertical field of view angle of 90°, use the correct aspect ratio, and set the near and far plane to some suitable values such that the scene (which is centered around the origin and is somewhat smaller than a unit cube) is fully visible—the actual choice is not too important in our case. But remember that using very large z-ranges increases z-fighting problems. After that, you should finally be able to see the track.

(e) Camera Transformation 2 **[4 Points]**

In addition to being able to view the track from a fixed point of view, we now also want to let the camera drive inside one of the cars, looking into the direction of travel. To this end, inside the function drawScene (...), in scene 5 (if (scene == 5)), set viewMatrix to a look-at transformation that depends on the values height and angle1. (Of course, we want you to use the lookAt (...) function you programmed earlier.)

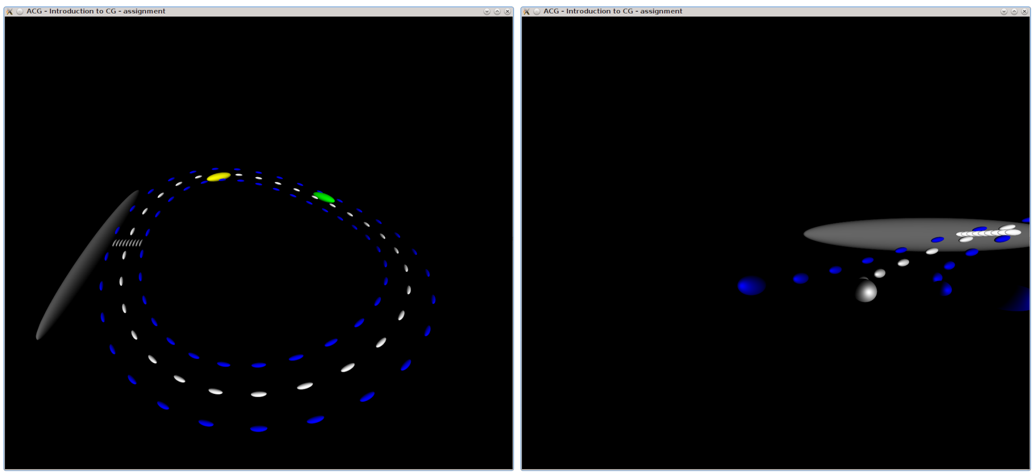

ACG - Introduction to CG - assignment
ACG - Introduction to CG - assignment

Figure 1: (a) The track seen with a FoV of 90° from the specified camera position. (b) Driving along the track.