University of Tübingen
Prof. Dr.-Ing. Hendrik P.A. Lensch
Chair of Computer Graphics
Faezeh Zakeri (faezeh-sadat.zakeri@uni-tuebingen.de)
Lukas Ruppert (lukas.ruppert@uni-tuebingen.de)
Raphael Braun (raphael.braun@uni-tuebingen.de)
Andreas Engelhardt (andreas.engelhardt@uni-tuebingen.de)
Philipp Langsteiner (philipp.langsteiner@student.uni-tuebingen.de)

01. November 2022

Computer Graphics
Assignment 2

Submission deadline for the exercises: 08. November 2022, 0:00

Source Code Solutions

- Upload **only** the src folder and all the files it contains, plus the CMakeLists.txt and files that are already exist in the exercise.zip. Do NOT include your build folder, results and exercise sheet in the zip folder you upload.
- Zip them first and upload **.**zip folder on Ilias.

Written Solutions

Written solutions have to be submitted digitally as one PDF file via Ilias.

2.1 Homogeneous Coordinates (5 + 5 = 10 Points)

a) Which of the following transformations can be done **without** homogeneous coordinates? Show how the transformation matrix would look like in 2D for all examples. We expect your answers to be in a similar form to the table below. (Note, a transformation of 2D coordinates in homogeneous form will be a 3×3 matrix)

	possible	**Matrix**
Scale		
Rotate		
Translation		
Shear		
Perspective		

b) In the example below, we apply various transformations without homogeneous coordinate. Convert the example to homogeneous coordinate, all transformations should be matrices. Multiply all matrices to get a final matrix representing a compound transformation.

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} + \begin{pmatrix} \cos(\frac{\pi}{4}) & -\sin(\frac{\pi}{4}) \\ \sin(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \end{pmatrix} \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

All number are real numbers ($\in \mathbb{R}$).

2.2 Vector Multiplication (5 + 10 = 15 Points)

Let $\vec{v} = \begin{pmatrix} 2 \\ 5 \\ 4 \end{pmatrix} \in \mathbb{R}^3$, $\vec{w} = \begin{pmatrix} 5 \\ 1 \\ 5 \end{pmatrix} \in \mathbb{R}^3$ and $\vec{x} = \begin{pmatrix} -4 \\ 0 \\ 2 \end{pmatrix} \in \mathbb{R}^3$ be three vectors.

a) Check whether the vectors are pairwise perpendicular, or whether the angle between them is acute ($< 90°$) or obtuse ($> 90°$) using the dot product.

b) Assuming $\vec{v}$ and $\vec{w}$ form a triangle starting at the origin. Calculate its normal vector and area (you can round to four decimal places). (Hint: normal vectors have to be normalized.)

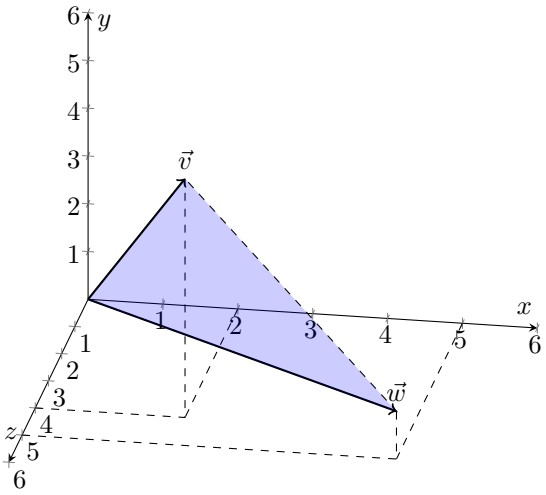

Figure 1: Vector $\vec{v}$ and $\vec{w}$ forming a triangle.

2.3 Space Transformations (10 Points)

Consider the vector space $\mathbb{R}^2$ which consists of vectors in two-dimensional x, y space such as

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \tag{1}$$

The most simple basis can be the unit vectors $\vec{e_1}, \vec{e_2}$ in x and y directions. We can express any vector in $\mathbb{R}^2$ as a linear combination of these basis vectors such as

$$\vec{v} = v_1 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + v_2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \text{or} \qquad \vec{v} = v_1\vec{e_1} + v_2\vec{e_2} \tag{2}$$

We could also express vector $\vec{v}$ in terms of some other basis space. Let's assume another pair of vector basis $\vec{u_1} = \binom{1}{2}$ and $\vec{u_2} = \binom{3}{3}$. Now, we can write our $\vec{v}$ as

$$\vec{v} = v_1'\vec{u_1} + v_2'\vec{u_2} \tag{3}$$

v_1' and v_2' are our new coordinates in the new basis space. While changing the basis, we are not affecting the vector $\vec{v}$ as it stays as before. Knowing this, find the coefficients of the mapping matrix M that relates coordinates of our standard basis $\vec{e_1}, \vec{e_2}$ to the coordinates of the new basis space.

$$M \begin{pmatrix} v_1' \\ v_2' \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \tag{4}$$

2.4 Orthonormal Basis (2 + 3 = 5 Points)

a) Explain briefly what an orthonormal basis is and what the difference is to an orthogonal basis.

b) Give an example for an orthonormal basis in $\mathbb{R}^3$ and prove that it is one.

2.5 Ray Intersection (10 Points)

The unit sphere can be parameterized by $x^2 + y^2 + z^2 = 1$, with $x, y, z \in \mathbb{R}$. Let $p = (4, 2, 6) \in \mathbb{R}^3$ be a point and $\vec{d} = \begin{pmatrix} -2 \\ -1 \\ -2.5 \end{pmatrix} \in \mathbb{R}^3$ a direction vector. Calculate the two intersection points between the ray $p + t \cdot \vec{d}$ ($t \in \mathbb{R}$) and the unit sphere.

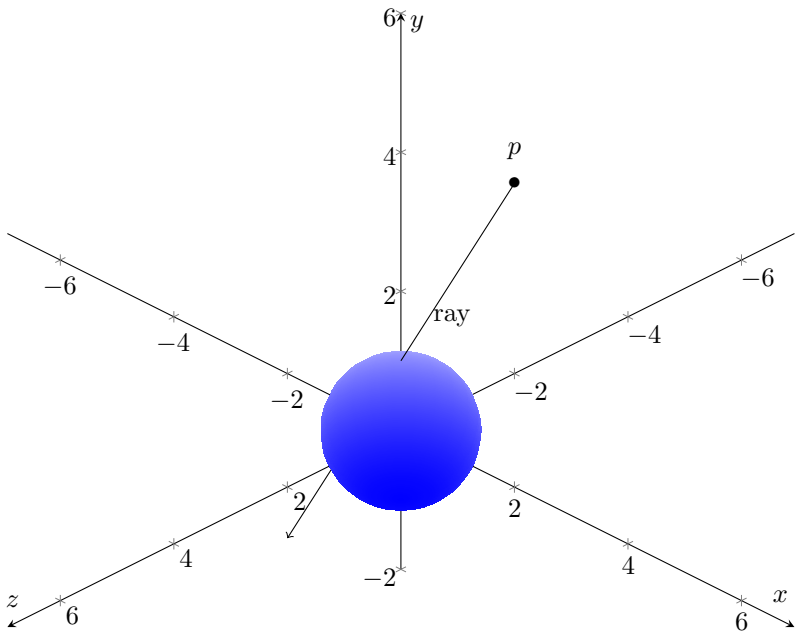

Figure 2: The unit sphere intersected by a ray.

2.6 Coding Exercise (10 + 30 + 10 Points)

In this exercise, we will deal with ray intersections and camera projections. Implement everything within `src/exercise02.cpp`. Again, there should be no need to touch any of the other files.

a) Implement the intersection test for `Rays` and Axis-Aligned-Bounding-Boxes (`AABBs`):

```
static bool intersect(const AABB& aabb, const IntersectionRay& ray);
```

Return `true` if the ray intersects the bounding box, and `false` otherwise.

Note that the component-wise inverse of the ray direction is available in the helper class `IntersectionRay` that you're already given.

If you want to try implementing the triangle intersection first, you can always `return true` here. Then, every ray will be tested against the entire mesh. This will be a lot slower!

b) Implement the intersection tests for `Rays` and `Triangles`:

```
static Intersection intersect(const Triangle& triangle,
                              const IntersectionRay& ray);
```

If there is no intersection, the value `t` of the `Intersection` `struct` needs to remain at ∞.

If there is an intersection, set the `intersection_point`, distance `t` and the surface `normal` at the intersection point by computing the geometric normal from the `Triangle`'s vertices.

c) Implement the ray generation for an orthographic camera:

```
static Ray orthographic(const Point2D& pos, const Point3D& origin,
                        const CameraFrame& cameraFrame,
                        const CameraParameters::Orthographic& orthographic);
```

The given `pos` is already given in normalized screen coordinates $\in [-1, 1]^2$:

$$\begin{pmatrix} \texttt{pos}_x \\ \texttt{pos}_y \end{pmatrix} = \begin{pmatrix} 2 \cdot (\texttt{pixel}_x / \texttt{res}_x - 0.5) \\ 2 \cdot (\texttt{pixel}_y / \texttt{res}_y - 0.5) \end{pmatrix} \tag{5}$$

The top-left corner corresponds to $(-1, -1)^T$ and the bottom-right corner corresponds to $(1, 1)$.

The output should respect the `left`, `right`, `top`, and `bottom` parameters given in the `Orthographic` `struct`. I.e., the `origin` of your `Ray` should deviate from the given `origin` of the `Camera` by [`left`, `right`] towards the `right` vector and [`bottom`, `top`] towards the `up` vector in world-space.

Since the image starts in the top-left corner, the `up` vector points towards $-y$ in terms of pixels.

Tracing rays in "Debug" mode is very slow. Consider using the "Release" or "RelWithDebInfo" mode for these tasks. To test your implementation, simply make sure that you render the same image on the right with your ray tracer as the OpenGL preview computes on the left.

You can play around with the camera parameters using the toolbox in the middle. To update the ray traced image on the right, click the "(Re-)render" button. Note that the "Display Options" do not influence the ray traced image. You can also zoom into the image to inspect individual pixel values. To reset the 2D image view, click the "Reset View" button.

You can see an example of the application in Figure 3.

Figure 3: The GUI of the demo application with the OpenGL 3D view on the left and the ray traced image on the right.