Prof. Dr.-Ing. Ralf Steinmetz
Multimedia communications Lab

Dr. Florian Mehm
Dipl. Inf. Robert Konrad

TECHNISCHE
UNIVERSITÄT
DARMSTADT

„Game Technology“
Winter Semester 2016/2017

Solution 3

General Information

- The exercises may be solved by teams of up to three people.
- The solutions have to be uploaded to the Git repositories assigned to the individual teams.
- **The submission date (for practical and theoretical tasks) is noted on top of each exercise sheet.**
- If you have questions about the exercises write a mail to game-technology@kom.tu-darmstadt.de or use the forum at https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=557

1. **Practical Tasks: Triangle Mesh Rendering (5 Points)**

Implement a basic software triangle renderer. Specifically, implement keyboard camera controls by carrying out rotation, translation and projection as described in lecture 3. You may use the Kore::vec3/4 classes to store data, but not the Kore matrix or quaternion classes. As an ungraded bonus, you can try out mouse camera control.

You will find comments in the code that direct you to the correct positions to add your code.

The suggested order is to start with translation first, then the perspective transform, and finally rotations. Note the effect each transformation has on the result. In case the model is not visible anymore after a change you made, check your calculations with some example data, e.g. the point (0, 0, 0), and find out at which position on the screen it ends up in.

https://github.com/TUDGameTechnology/Exercise3.git contains code for triangle rasterizing and mesh loading and a mesh to get you started. You can either copy the code changes manually or just pull them into your own repository using
git pull https://github.com/TUDGameTechnology/Exercise3.git

Additional Information:

- A video showing the solution for this exercise is available on the website.
- There was a typo in the lecture slides, on slide 25 in line 6, "… + sin(camera.ry) * dz;" should be "… - sin(camera.ry) * dz;" This might have led to some of your rotations behaving unexpectedly.
- The model can end up either very small or very much distorted due to the perspective. In the solution shown in the video, the image is scaled up before drawTriangle is called. To achieve this, you can use code like the following:

```
float drawScale = 512.0f;
drawTriangle(
       x1 * drawScale + width / 2, y1 * drawScale + height / 2,
       x2 * drawScale + width / 2, y2 * drawScale + height / 2,
       x3 * drawScale + width / 2, y3 * drawScale + height / 2);
```

Please remember to push into a branch called "exercise3".

You can find the source code for the solution at https://github.com/TUDGameTechnology/Solution3.

2. Theoretical Tasks (5 Points)

2.1 Transformations (2 Points)

a) In the general case, is the order in which only translations are applied to an object relevant? Explain in your own words why or give a counter-example.

Translations are vector additions, and this is a commutative operation. No matter in which order we add up the vectors, we end up at the same position.

b) In the general case, is the order in which any combination of transformations (translation, scale, rotation, shearing) is applied to an object relevant? Explain in your own words why or give a counter-example.

As a counter-example, we use the combination of scaling and translating a vector.

If we use $v_1 = \begin{pmatrix}1\\2\\3\end{pmatrix}$, $v_2 = \begin{pmatrix}1\\1\\1\end{pmatrix}$ *and* $s = 5$, *we get*

$$s \cdot (v_1 + v_2) = 5 \cdot \begin{pmatrix}1+1\\2+1\\3+1\end{pmatrix} = 5 \cdot \begin{pmatrix}2\\3\\4\end{pmatrix} = \begin{pmatrix}10\\15\\20\end{pmatrix}$$

$$s \cdot v_1 + v_2 = 5 \cdot \begin{pmatrix}1\\2\\3\end{pmatrix} + \begin{pmatrix}1\\1\\1\end{pmatrix} = \begin{pmatrix}5\\10\\15\end{pmatrix} + \begin{pmatrix}1\\1\\1\end{pmatrix} = \begin{pmatrix}6\\11\\16\end{pmatrix}$$

We see that the results differ.

Geometrically, we can observe that scale (unless we scale by 0 or 1), shearing (unless the shearing parameters are 0) and rotation (unless we rotate by 0 degrees) change the space we are calculating in. For example, after scaling by a certain factor s, we operate in a space where all vectors are scaled by the factor s. Figure 1 illustrates this effect. Keeping this in mind, we can also geometrically understand why the order of operations is important.

Note: Several answers claimed that matrix multiplication is commutative. This is not true in the general case. As a counter-example, consider the matrices we can construct for example from the scale and translation above. As we have seen, the order for these operations matters, therefore, the equivalent matrices will also differ.

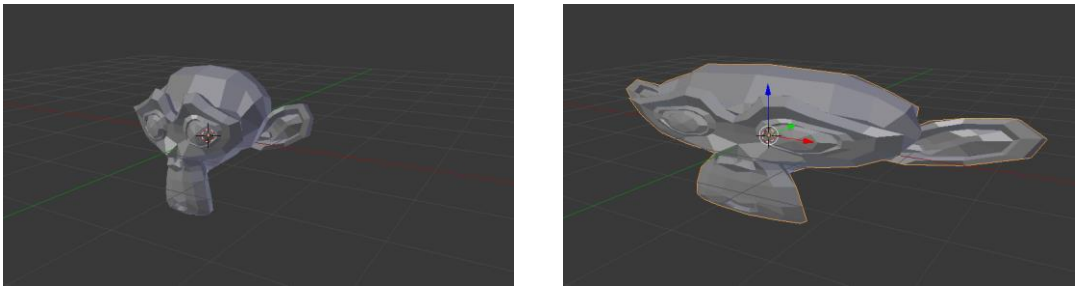

Figure 1: Left: Before applying a scale of 2.5 along the x-axis. Right: After applying the scale. All further operations will operate in a space that is scaled along the x-axis with a scale factor of 2.5.

2.2 Angle calculations (2 Point)

You are writing a top-down game in which characters are playing with water guns. You want to write a function to calculate the angle between where a character A is currently facing and where a potential target character B is. The game happens in the X-Z-plane, and Y is up.

A is currently at position (0, 0, 0) and is oriented so that its forward-direction is the normal vector (1, 0, 0). The target is at position (2, 0, -3). Calculate the angle between A's forward direction and the direction towards the target B. Show your calculations as part of the answer.

We can use the dot product. We know that

$$a \cdot b = \cos(\alpha)\,|a||b|$$

We divide by the lengths and get

$$\cos(\alpha) = \frac{a \cdot b}{|a||b|}$$

Therefore,

$$\begin{aligned}\alpha &= \cos^{-1}\left(\frac{a \cdot b}{|a||b|}\right)\\ &= \cos^{-1}\left(\frac{2}{1 \cdot 3.61}\right)\\ &= \cos^{-1}(0.55)\\ &= \mathbf{0.982793}\end{aligned}$$

*In degrees, this equals an angle of **~56.31 degrees**.*

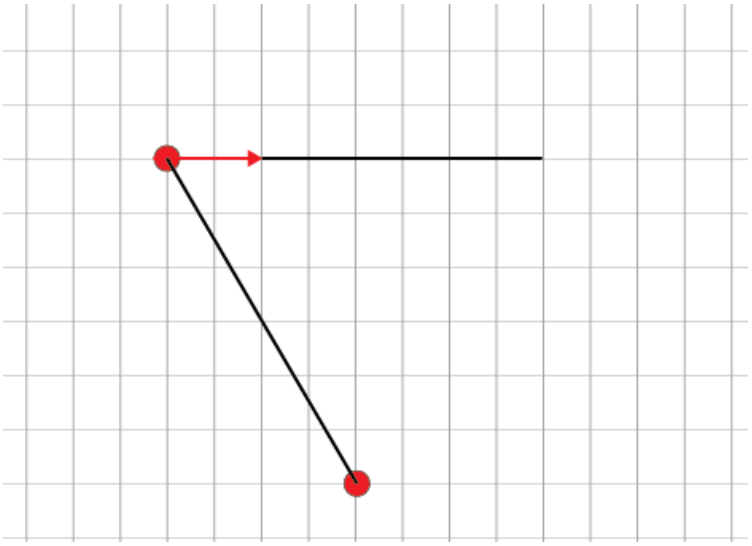

Figure 2: The situation from above

2.3 Raytracing and Rasterization (1 Point)

Explain in your own words the differences between raytracing and rasterization.

Raytracing

- *We send out a ray for each pixel of the image*
- *We color the pixel by following the ray and calculating the color at which it intersects the world*

Rasterization

- *We transform from 3D to 2D*
- *We draw primitives (triangles) in 2D*
- *We fill the 2D image by drawing each point where the triangle is visible*