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 4

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: Textures and Depth Buffers (5 Points)**

Extend your software renderer to support texture mapping and depth buffering. For closer instructions see the comments in the `shadePixel` function in the source code.

https://github.com/TUDGameTechnology/Exercise4.git contains code for texture coordinate loading and interpolation. You can either copy the code changes manually or just pull them into your own repository using https://github.com/TUDGameTechnology/Exercise4.git

Please remember to push into a branch called "exercise4".

You can find the solution for the practical task at https://github.com/TUDGameTechnology/Solution4.git.

2. Hypertheoretical Task: Matrix Multiplication Performance (5 Points)

2.1 Weird things (1 Point)

In lecture 4, the three problems listed below were addressed. Choose one of them and describe both the problem and the possible solution in your own words.

Weird depth problems
Weird textures
Weird rotations

Weird depth problems

- *Our result depends on the depth order in which we painted*
- *Solution: Use a z-Buffer, only write the color if it is in front of the front-most pixel painted so far*

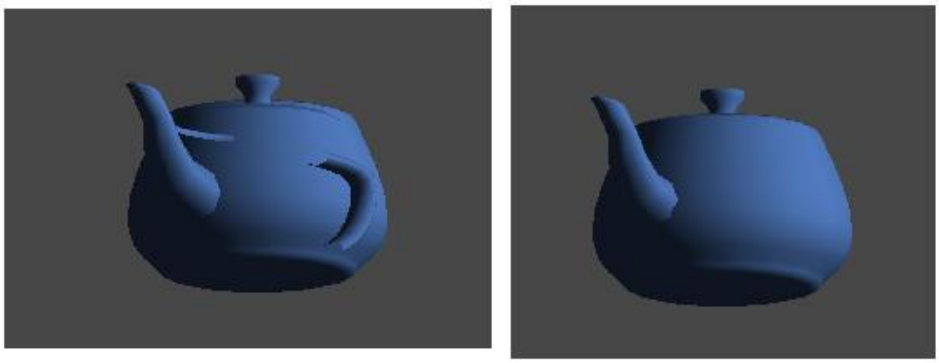

Figure 1 Left: Teapot rendered without z-buffer. Right: Rendered with z-buffer. (http://forum.unity3d.com/threads/problem-when-rendering-a-3d-module-with-z-buffer-disabled-ztest-always.143075/)

Weird textures

- *The textures are sampled equally regardless of the perspective (we're doing a 2D operation where a 3D operation would be appropriate)*
- *We can fix it by perspective correct texture mapping.*

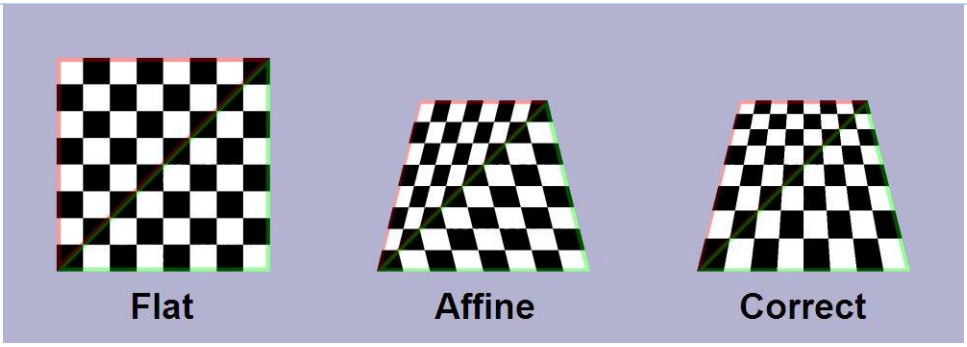

Figure 2: The effect of perspective correct texture mapping (https://en.wikipedia.org/wiki/Texture_mapping#/media/File:Perspective_correct_texture_mapping.jpg)

Weird rotations

- *Rotations with Euler angles are weird to interpolate*
- *We can have gimbal lock*
- *Solution is to use quaternions*

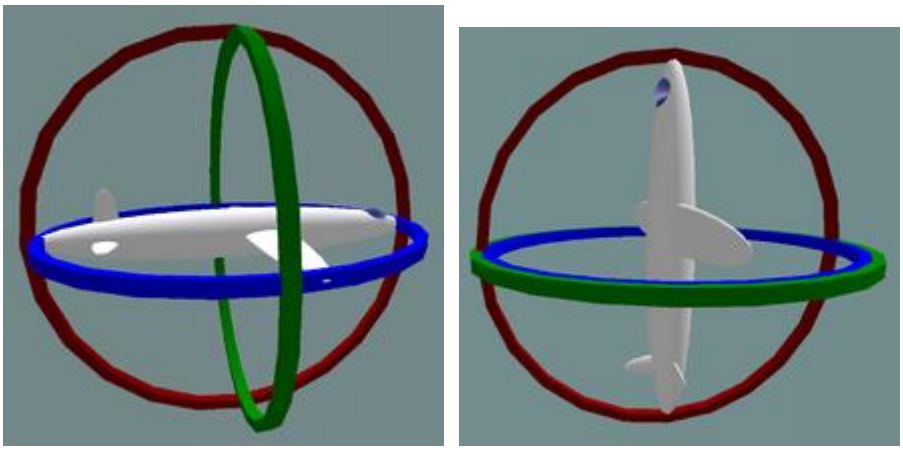

Figure 3: Gimbal lock: After one rotation, two of the gimbals are in the same plane, so one degree of freedom is lost, since rotating around two different axes will result in the same rotation. (https://en.wikipedia.org/wiki/Texture_mapping#/media/File:Perspective_correct_texture_mapping.jpg)

2.2 Matrix analysis (2 Points)

a) What geometric operation(s) does this matrix encode? How can you tell?

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We multiply the matrix to a general vector. We get the following result:

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} 2x+1 \\ 2y \\ 2z \\ 1 \end{pmatrix} = \begin{pmatrix} 2x \\ 2y \\ 2z \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The vector is scaled uniformly by the factor 2, and is translated by one unit in the x-direction.

b) What geometric operation(s) does this matrix encode? How can you tell?

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Note: The matrix was intended to be of the following form:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Only one group noticed this in their answer, but most groups were correct in assuming that perspective projection was intended.

We multiply the matrix to a general vector. We get the following result:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z \end{pmatrix}$$

We get a matrix with w replaced by z. If we turn this into a homogenous coordinate again by dividing by the fourth row, we get:

$$\begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \\ 1 \end{pmatrix}$$

This vector is a very simple perspective projection.

2.3 Alphabet war (1 Point)

Your game engine's rasterizer uses a z-buffer. You want to display decals on objects. You implement them by creating quadrilaterals which are exactly aligned with the target surface and which have a distance of 0 to the surface (i.e. like gluing a poster to a wall).

Your game shows weird artifacts where the surface and the decal are both partly visible. What happened and what could be a fix?

The resulting effect is referred to as z-fighting. When two surfaces share the same depth, there can be problems due to floating point inaccuracies.

A simple fix could be to apply the decal with some distance to the target surface that is larger than the inaccuracies, or to render it after the surface has been drawn without depth testing (but only if we know that the decal is not obstructed by something else).

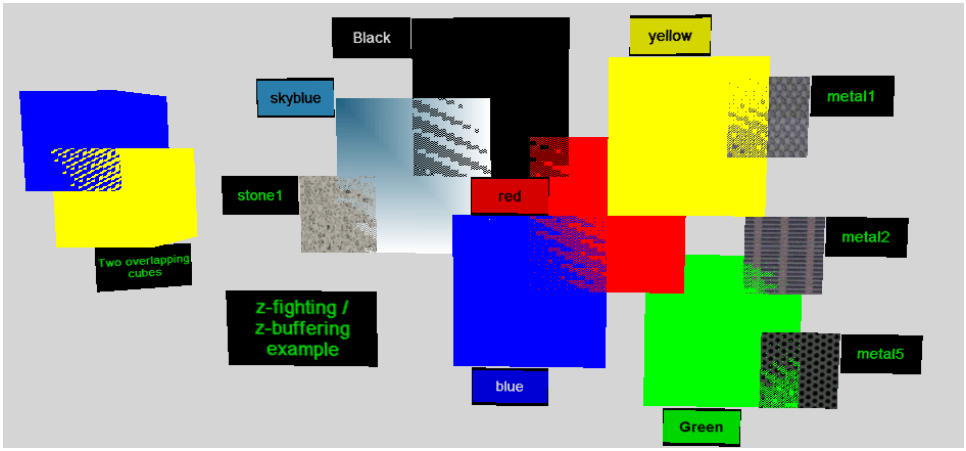

Figure 4: Examples of z-fighting (https://upload.wikimedia.org/wikipedia/commons/5/5f/ZfightingCB.png)

2.4 Performance (1 Point)

Which is more performant: Phong or Gouraud-Shading? Explain in your own words why.

The bulk of the calculation of Gouraud-Shading is done in the vertex stage (of which we have often considerably less than fragments), and the result is interpolated across the surface.

In Phong-Shading, the bulk of the calculation lies in the pixel stage, which is called more often than the vertex stage. Therefore, Phong shading is more expensive than Gouraud.