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

Dipl. Inf. Robert Konrad
Polona Caserman, M.Sc.

TECHNISCHE UNIVERSITÄT DARMSTADT

„Game Technology“
Winter Semester 2017/2018

Solution 5

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

P5 Practical Tasks: Blinn-Phong lighting (5 Points)

Implement the Blinn-Phong shading model in GLSL. Also port your camera code to GLSL. Define the camera parameters using uniform declarations and set their values using Kore's Graphics API.

Feed the information about the camera to your shader by adding code to Exercise.cpp. Implement shader.vert and shader.frag accordingly (carry out transformations in the vertex shader, carry out shading in the fragment shader).

https://github.com/TUDGameTechnology/Exercise5.git contains additional code to help you out. You can either copy the code changes manually or just pull them into your own repository using
git pull https://github.com/TUDGameTechnology/Exercise5.git

Please remember to push into a branch called "exercise5".

You can find the solution for the practical task at https://github.com/TUDGameTechnology/Solution5.git.

T5 Theoretical Tasks: Where there is light, there must be shadow (5 Points)

T5.1 Blending Order (2 Points)

According to our lecture, rendering semitransparent objects correctly requires correct rendering order (from back to front). Have a closer look at the equations for standard and additive blending and verify or debunk our statement for both equations individually.

Standard blending

For standard blending, we use:

$$f(s,d,\alpha) = \alpha \cdot s + (1-\alpha) \cdot d$$

For two semitransparent objects:

$$\begin{aligned} & f(s_1, f(s_2, d, \alpha_2), \alpha_1) = \alpha_1 \cdot s_1 + (1-\alpha_1) \cdot f(s_2, d, \alpha_2) \\ & = \alpha_1 s_1 + (1-\alpha_1)(\alpha_2 s_2 + (1-\alpha_2)d) \\ & = \alpha_1 s_1 + (1-\alpha_1)(\alpha_2 s_2 + d - \alpha_2 d) \\ & = \alpha_1 s_1 + \alpha_2 s_2 + d - \alpha_2 d - \alpha_1 \alpha_2 s_2 - \alpha_1 d + \alpha_1 \alpha_2 d \end{aligned}$$

If we exchange $\alpha_1, \alpha_2, s_1, s_2$ we arrive at:

$$f(s_2, f(s_1, d, \alpha_1), \alpha_2) = \alpha_2 s_2 + \alpha_1 s_1 + d - \alpha_1 d - \alpha_2 \alpha_1 s_1 - \alpha_2 d + \alpha_2 \alpha_1 d$$

$$\begin{aligned} & f(s_1, f(s_2, d, \alpha_2), \alpha_1) - f(s_2, f(s_1, d, \alpha_1), \alpha_2) \\ & = \alpha_1 s_1 + \alpha_2 s_2 + d - \alpha_2 d - \alpha_1 \alpha_2 s_2 - \alpha_1 d + \alpha_1 \alpha_2 d \\ & + \alpha_2 s_2 - \alpha_1 s_1 - d + \alpha_1 d + \alpha_2 \alpha_1 s_1 + \alpha_2 d - \alpha_2 \alpha_1 d \\ & = \alpha_1 \alpha_2 (s_1 - s_2) \end{aligned}$$

We see that, in the general case, switching the order results in a difference.

Additive blending

For additive blending, we use:

$$f(s,d,\alpha) = \alpha \cdot s + d$$

For two semitransparent objects:

$$f(s_1, f(s_2, d, \alpha_{2,}), \alpha_1) = \alpha_1 \cdot s_1 + f(s_2, d, \alpha_2)$$
$$= \alpha_1 s_1 + \alpha_2 s_2 + d$$

If we exchange $\alpha_1, \alpha_2, s_1, s_2$, we arrive at:

$$f(s_2, f(s_1, d, \alpha_{1,}), \alpha_2) = \alpha_2 s_2 + \alpha_1 s_1 + d$$

$$f(s_1, f(s_2, d, \alpha_{2,}), \alpha_1) - f(s_2, f(s_1, d, \alpha_{1,}), \alpha_2)$$
$$= \alpha_1 s_1 + \alpha_2 s_2 + d - \alpha_2 s_2 - \alpha_1 s_1 - d$$
$$= 0$$

We see that the order is not important.

Note: Some groups constructed counter-examples of the form $f(s, d, \alpha) \neq f(d, s, \alpha)$. Simply switching the arguments like this is not a valid counter-example, since this is actually a different operation. In practice, we have rendered something non-transparent and then render several semi-transparent objects over the background, therefore, we need to show $f(s_1, f(s_2, d, \alpha_{2,}), \alpha_1) \neq f(s_2, f(s_1, d, \alpha_{1,}), \alpha_2)$.

Example

The following images have been created in a graphics program to show the two operations

Normal blending

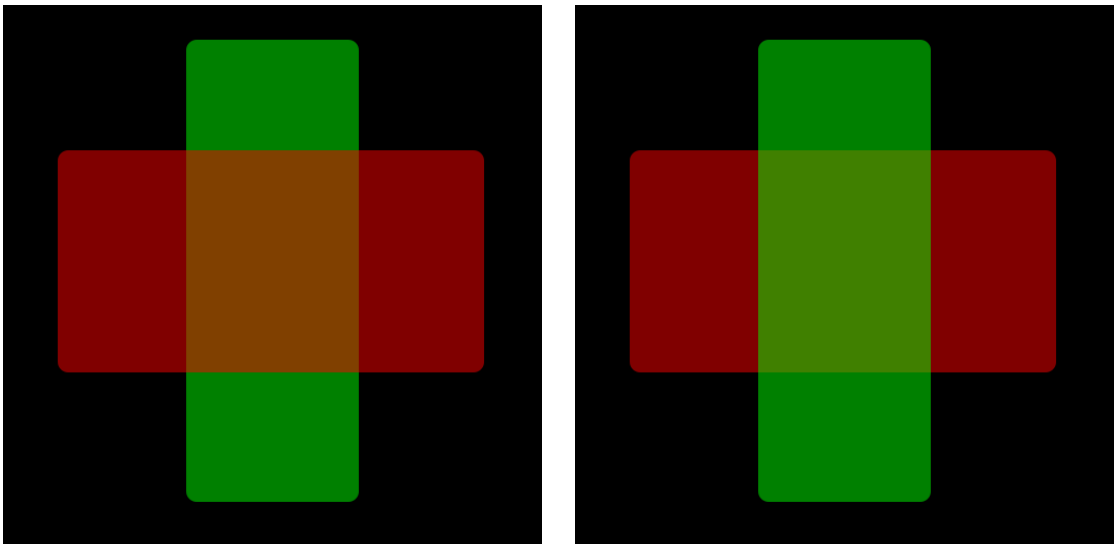


Figure 1: Standard blending - Left: Red object on top, Right: Green object on top

Additive blending

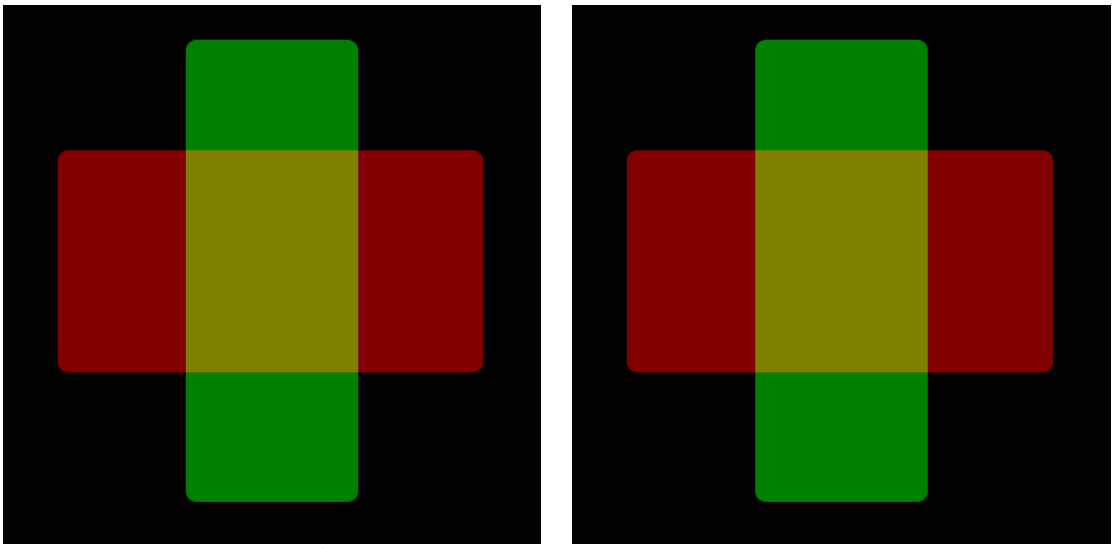


Figure 2: Additive blending - Left: Red object on top, Right: Green object on top

T5.2 Antialiasing (2 Points)

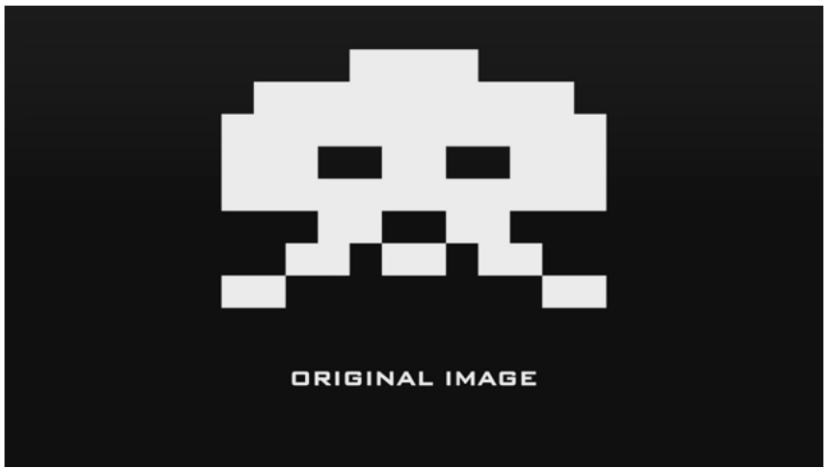
What is the major shortcoming of post-process anti-aliasing? Why is it used nonetheless?

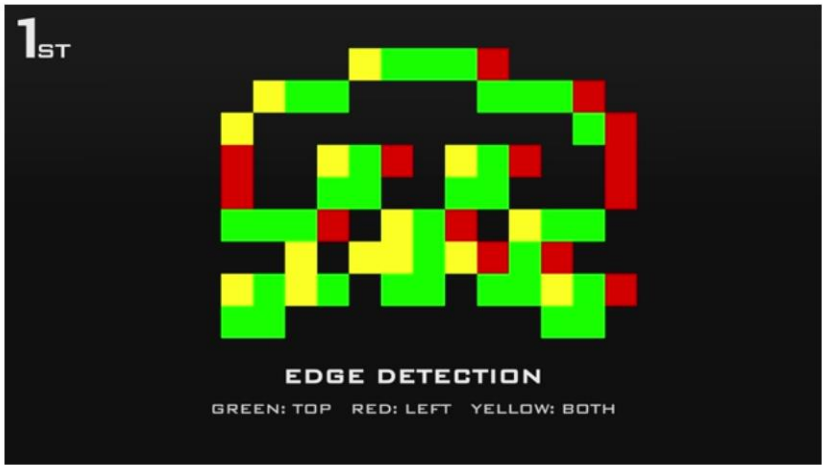
Post-process antialiasing happens after the image has been rendered. In essence, post-process AA results in blurring, since it can't get higher resolution that that at which the image was rendered. Nevertheless, it is used, since it can be cheaper than other options and might be integrated into a post-processing step that is done anyway in the engine.

An example of this is "MLAA", Morphological Anti-Aliasing:

"The algorithm detects borders (either using color or depth information) and then finds specific patterns in these. Anti-aliasing is achieved by blending pixels in the borders intelligently, according to the type of pattern they belong to and their position within the pattern. Pre-computed textures and extensive use of hardware bilinear interpolation to smartly fetch multiple values in a single query are some of the key factors for keeping processing times at a minimum."

See http://www.iryoku.com/mlaa/ for more information on this algorithm.





T5.3 Roughness (1 Point)

In the literature, diffuse reflection can denote two things. In the lecture we introduced diffuse reflections as light that penetrates the molecular structure of an object and then leaves in a completely random direction. Elsewhere diffuse reflection is defined as direct reflections from rough surfaces. In reality these semi-diffuse reflections look like blurred, direct reflections. In the lecture we showed two special kinds of texture maps, mip maps and cube maps. How can those be used to implement blurred reflections aka roughness?

Cube maps can be used to create reflections. A cube map is a texture that includes 6 sides of a cube, to capture all of the surroundings of an object.

Mip maps are increasingly smaller versions of the same image. If we switch between mip levels while staying at the same distance to an object, the texture will be more and more blurry, since we are effectively enlarging a texture that is getting smaller to draw the texture.

As a combination, we could choose a mip-level of the cube map that corresponds to the roughness of the surface we are trying to emulate while we are calculating reflections, in order to get a blurry reflection.

Note that simply generating the mip levels by downsampling the original image is not physically correct. For information how this preprocessing step is done in Unreal Engine 4, see the course notes by Brian Karis at

SIGGRAPH 2013's "Physically Based Shading in Theory and Practice" course:
http://blog.selfshadow.com/publications/s2013-shading-course/#course_content

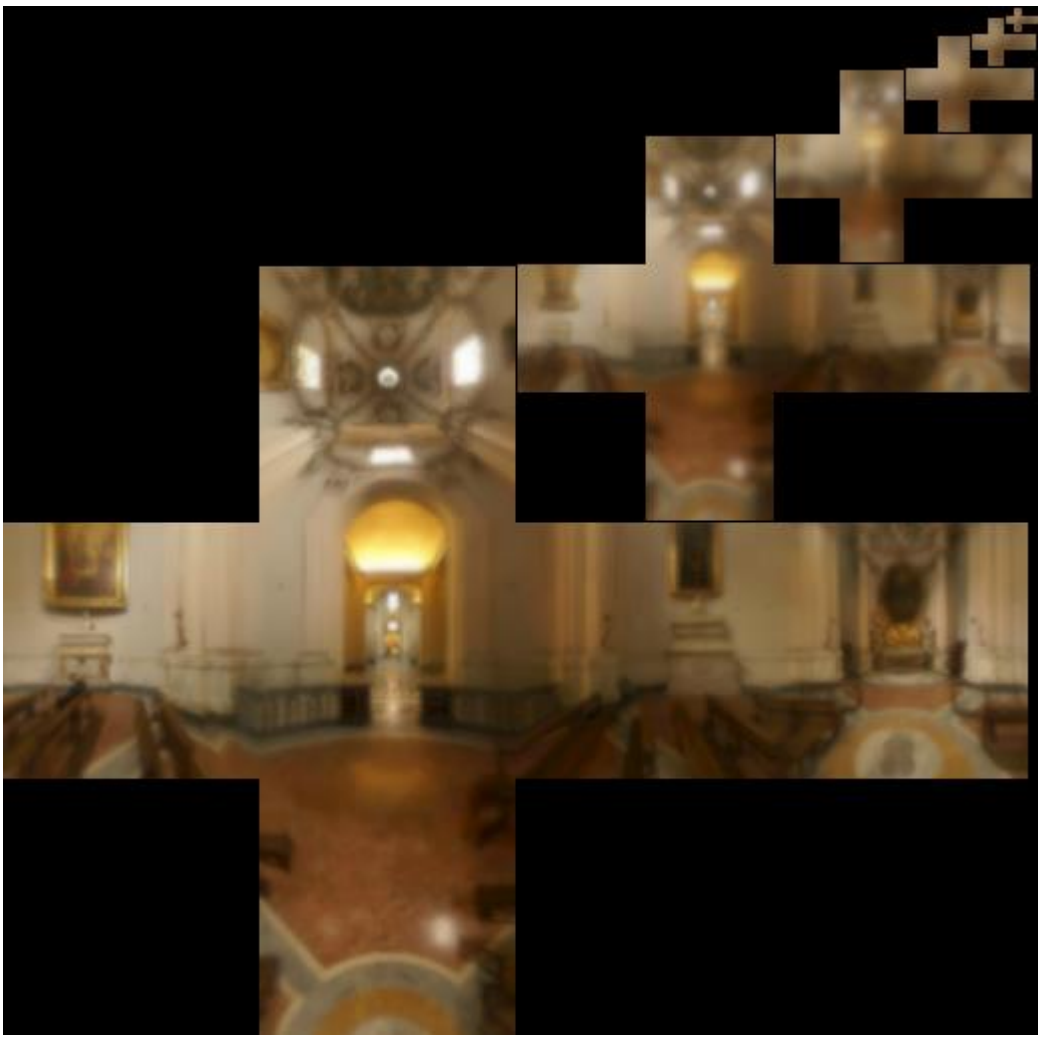


Figure 3: An example for cube map mip map levels (http://jmonkeyengine.org/301308/physically-based-rendering-part-3/)