# Praktikum Echtzeit-Computergrafik
# Assignment 10 – Transparency Rendering

Technische Universität München
Institut für Informatik
Lehrstuhl für Computergrafik & Visualisierung

Christoph Neuhauser, Simon Niedermayr – SS 24

*In this assignment, you will play around with transparency rendering in ShaderLabWeb (https://vmwestermann10.in.tum.de/) and learn more about the different transparency blending modes.*

# Introduction

# Tasks

## Task 1

Read the following article giving an introduction into transparency rendering: https://developer.nvidia.com/content/transparency-or-translucency-rendering

You may skip the parts on "Reduced Rate Blending", "Transmittance Thresholding" and "Weighted, Blended Order-Independent Transparency (OIT)".

## Task 2

Let's assume we have three colored planes as shown in the figure below and the background has the color $(0, 0, 0, 1)^T$. Compute the final color for the view ray using alpha blending with the OVER and UNDER operator as described in the article above.

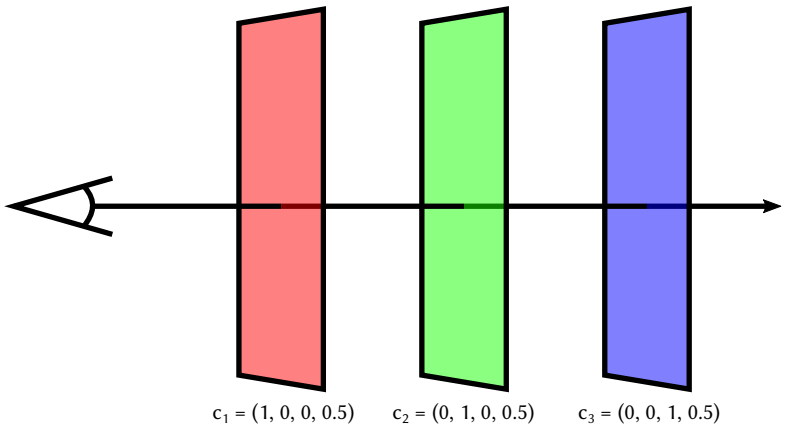


Figure 1: Colored, transparent planes. Colors are specified as straight (i.e., not pre-multiplied) alpha.

## Task 3

In this task, we want to create a mesh containing three planes similar to what was used in the last exercise.

- Write a Java program which creates an .obj mesh[1] with three planes. You do not need to store normals or texture coordinates. It is recommended to create quads with a bounding box of $[-4, 4] \times \{d_i\} \times [-4, 4]$ with $d_i \in \{0, 4, 8\}$.
- Load the mesh in ShaderLabWeb. Pass `vertexPosition`, i.e., the model space vertex position, to the fragment shader. Color the vertices depending on their y coordinate with the three colors specified in the figure in Task 2.

After loading the mesh in ShaderLabWeb, try to answer the questions below.

1. Set the correct settings for the OVER operator with straight (i.e., not pre-multiplied) alpha in the `Model` tab. What settings do you need?
2. Turn the object around. As you will see, the planes do not always appear as expected, but we only see the closest plane when looking at them from one side. What value do you need to set for `Model` → `Depth Test` to solve this problem? Why is that necessary?
3. When using `Model` → `Depth Test` → `Disabled`, you can notice that the blending results are still wrong when looking at them from one specific side. Why is this happening? What do you think can be done to solve this problem?
4. What settings do you need to change in `Model` to get additive rendering, i.e., the resulting color is the sum of the colors of all fragments falling into the pixel?

[1] https://en.wikipedia.org/wiki/Wavefront_.obj_file

## Solution: Task 2

$c_{final} = (0.5, 0.25, 0.125, 1)^T$

## Solution: Task 3

1. The correct factors are:

   srcColor: SRC_ALPHA, dstColor: ONE_MINUS_SRC_ALPHA, srcAlpha: ONE, dstAlpha: ONE_MINUS_SRC_ALPHA. Please note that other factors for the alpha values may result in the same color, as the background color is 100% opaque black.

2. Answer: The problem is that the depth test makes sure that fragments are discarded when they lie behind a previously rendered fragment that is closer to the viewer. However, for transparency rendering, this is something we don't want, as one can potentially see further fragments lying behind a transparent fragment. Consequentially, transparent geometry is usually rendered after opaque geometry with the depth test set to `Disabled` (`Always` has the same effect).

3. The problem is that the OVER and UNDER operator are order dependent. This means that for the OVER operator, which we set in 1., the plane geometry (i.e., the triangles) must be sorted from front to back, while for the UNDER operator it is the other way around. A solution to this problem is to either sort the geometry in the CPU side code every frame before rendering, or using an order-independent transparency algorithm. You can learn more about order-independent transparency techniques in the course *Image Synthesis* offered by our chair for Master students. For ShaderLabWeb, the only solution is to feed it a mesh with the correct plane order (i.e., back-to-front) for the side we view them from, as we do not have access to the CPU side code of it as a user to sort them at runtime.

4. We can use SRC_ALPHA and ONE for the source and destination color and alpha blending factors.