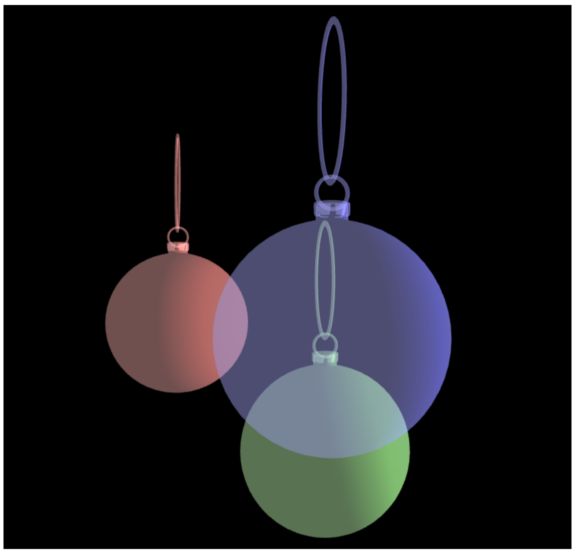
Prof. Dr. Markus Gross

Visual Computing

Exercise 10: Shaders & Blending

Hand-out Date: 26 November 2024



Goals

- Learning GLSL (Graphics Library Shader Language).
- Learning the difference between vertex and fragment shaders.
- Understanding depth test and alpha blending.

Resources

The lecture slides and exercise slides are accessible via the Visual Computing Course Web Page.

Tasks

In this exercise, we are going to render the same "Christmas decoration" scene as in previous exercises but we want to make the decorative balls semi-transparent! We break the task into 3 small subtasks as follows:

1. Preparation

To correctly render transparency, we need to render back to front with respect to the camera. In this first subtask, we will implement some helper functions that help with depth sorting in later steps.

(a) Rearrange vertex indices such that each ball's indices are consecutively grouped, and separated from each other. More specifically, store vertex indices of the three balls into three arrays and concat them into a new array to be `mesh.indices`. This step makes rendering a particular ball easier later. Please fill in the missing lines inside the function `rearrange(...)`. Hint: you can use the same strategy as in `calVertexColor`.

(b) Since we are only going to sort the three balls and we assume they are well separated, we can just compare their depth based on the positions of the ball centers. Complete the function `getDecorativeBallCenters(...)`, which returns the center positions of the three grouped vertices.

2. WebGL Blending

In the `render` function, enable blending with `gl.enable(gl.BLEND);`. Then, choose the correct blending function with `gl.blendFunc(...)`. Now the rendering pipeline should take into account the transparency of the objects (the alpha channel in the objects' color). Don't forget to change also the code in the vertex shader such that the alpha values are set as desired.

3. Depth Sorting & Rendering

The last step to render our semi-transparent scene is to draw the objects back to front.

(a) Complete the function `sortDecorativeBalls(...)` which takes as input the ball centers, the model view matrix, and returns the sorted indices. Note that the `z` coordinates of the transformed center positions correspond to the depth of the objects. Javascript has a built-in function `sort` for arrays.

(b) Draw the balls in the rendering pipeline, *i.e.*, call `gl.drawElements(...)` once for each ball. Thanks to 1(a), we can now render the corresponding triangles of each ball by setting the correct offset, given that they have the same number of vertices and the vertices are grouped consecutively. And now we should be able to see our semi-transparent "Christmas decoration" scene on the screen!

Discussions

Note that we made some important assumptions and simplifications in this exercise to render transparency. We assume the three balls are well separated. We sort only among them and draw them sequentially. This could be very expensive for complex scenes with many objects. Also, the back faces are ignored. Is this strictly correct? What if objects intersect each other, *e.g.*, the rings on top of each ball? On what level should the sorting be performed? Should we sort on an object level for the whole scene? Or should we sort all the triangles? What if two triangles intersect each other? How can we render transparency fast? For those who are interested, check out related work on order-independent transparency.