Homework 1 Gaussian Splatting

Žan Terplan

I. Introduction

Gaussian splatting is a technique used for rendering point clouds and volumetric data, where each point is represented by a Gaussian function. This method allows for smooth blending and efficient visualization, making it good for real-time rendering of 3D scenes. The goal of this assignment is to implement a point cloud renderer based on Gaussian splatting. We explored 3D transformations, rasterization and blending of splats, as well as challenges like perspective correction, order-based blending and Gaussian falloff.

II. Implementation

In this section, we will discuss the implementation of the key components that were used to render the splats correctly. Each subsection will describe a specific task of the project, including basic transformations, perspective scaling, blending with depth sorting and Gaussian falloff.

A. Basic transformations and point rendering

In this subsection, the goal is to load the splat data, apply 3D transformations and render the splats as basic points. Positions and colors from the splat data are loaded from a binary file and transformed into normalized coordinates. The camera transformation, based on a perspective projection and look-at matrix, is used to position the camera and objects in the scene. We successfully implemented a system where the splats are displayed on the screen as basic points using WebGL. We also implemented an orbit camera around the center of the splat when dragging the mouse in different directions (up, down, left and right). An additional implementation was the use of WASD keys for camera movement in the up, down, left and right directions. Scrolling using the mouse wheel was also incorporated as a zoom in/out functionality.

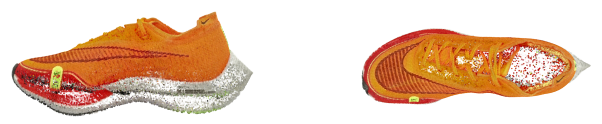
Figure 1: Left and top view of the final Nike shoe splat. There are visible holes because not many splats are present in that areas. That is corrected with the optional task, where you implement non-uniform scaling by using the rotation and scale data of the splats.

B. Perspective-correct scaling

Here, the goal is to correctly scale the splats based on their distance from the camera. The splats are rendered with a scale factor that is inversely proportional to their depth, ensuring that objects closer to the camera appear larger. The scaling factor is applied in the vertex shader and the size of each splat adjusts based on its depth. We've implemented the scaling logic and dynamic adjustment of the parameter s with the use of keys $+$ and $-$.

C. Order-correct blending

This subsection addresses sorting the splats by their depth (distance from the camera) to ensure that closer splats are rendered in front of farther ones. It also explains blending of the splats in the correct order, which is important to prevent incorrect color layering. We implemented a sorting function that orders the splats based on their depth in view space. We used this along with straight (non-premultiplied) alpha blending to blend them correctly.

D. Gaussian falloff

The goal here is to implement the Gaussian falloff function, where the alpha component of each splat fades out with distance from the center of the splat. We implemented the falloff in the fragment shader, where the alpha value is modified using a Gaussian function, dependent on the distance from the splat's center. The result is that splats blend more smoothly at their edges.

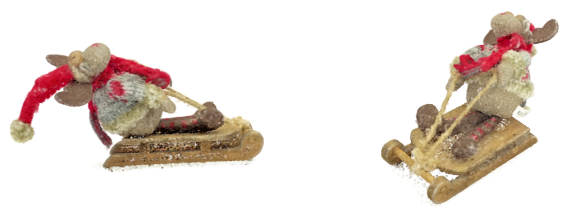
Figure 2: Right and diagonal view of the final plush splat. Some holes are again visible, but the overall structure looks familiar and correct.

III. Experiments

The performance of the rendering process was measured using two different approaches. We measured CPU timing by measuring time of the render() function, which handles clearing the canvas, setting up matrices, sorting splats by depth and rendering of the splats. It was executed 50 times on the shoe splat and the average CPU time was calculated at 432 ms. We've also displayed FPS in the bottom left corner of the screen. It averages around 2-2.5 FPS for the shoe and plush splats, and around 0.5 FPS for the train splat.

IV. Conclusion

In this assignment, we successfully implemented a point cloud renderer using Gaussian splatting. We explored 3D transformations, perspective scaling, depth-based blending and Gaussian falloff. Implementing each of these tasks contributed to completing the mandatory part of the assignment. Future improvements could involve implementing fast sorting techniques to enhance performance or non-uniform scaling to improve the model visually.