Homework 1
Gaussian splatting

Advanced computer graphics 2023/24

1 Introduction

In this homework, you will get familiar with the basic principles of Gaussian splatting [KKLD23]. Gaussian splatting is a method for rendering volumetric data, especially point clouds, where each point is associated with a Gaussian function. In addition to position, these Gaussian functions can have a scale and an orientation so that they can represent structures of various shapes and sizes.

The goal of this homework is to write a point cloud renderer based on Gaussian splatting. You will have to program the necessary 3D transformations, rasterization, and blending of the splats.

2 Input

The input to your program will be a binary file with a sequence of splats with the following properties: position ($3 \times$ float32), scale ($3 \times$ float32), color (RGBA with straight alpha, $4 \times$ uint8), rotation ($4 \times$ uint8 representing the components of a quaternion, each component c is decoded as $(c-128)/128$). Each splat is therefore 32 bytes. No header is present at the start of the file. The number of splats in the file is determined by the size of the file.

3 Tasks

3.1 Basic transformations and point rendering (2 pts)

You can assume that the splats in the input file are in world space. You can, however, decide to include an affine model transformation of your choice. To view the data set, you must place a camera into the scene. The camera is defined by a view transformation and a perspective transformation. You are free to choose the camera parameters (near/far plane, field of view) as you see fit.

Construct a joint transformation matrix and use it to transform each of the splat centers into screen space, then color the underlying pixel with the color of the corresponding splat. For now, you can ignore the alpha component, scale, orientation and depth of the splats.

3.2 Perspective-correct scaling (2 pts)

When drawing splats as single pixels, we are ignoring perspective projection. Splats near to the camera should be drawn larger. The correct scaling factor is $1/z$, where z is the depth in view space (before perspective division). Since at this point we do not know the initial size of a splat, we can introduce a scaling parameter s, so that the final scale is s/z. Your program must support dynamically adjusting the scaling parameter s. Update the program to render the splats as view-aligned squares with side length $2s/z$ pixels.

3.3 Order-correct blending (3 pts)

The order in which the splats are drawn is important. When multiple splat centers are projected onto the same pixel, the color of the splat that was drawn last overwrites the previous colors. Therefore, to achieve a correct rendering, the splats must be drawn in a consistent order. This can be achieved through sorting the splats according to the screen-space depth before rendering.

At this point, we can incorporate the alpha component of the splat colors. Splat colors are using straight (non-premultiplied) alpha, so we must multiply the RGB components with the alpha component before use. When rendering a single pixel, you are presented with two colors: the source color RGB_s, A_s (the color of the splat), and the destination color RGB_d, A_d (the current color in the image). To blend

sorting - po projective in view matrixu

them together, you can use one of the alpha blending approaches. In this assignment, we will initialize the image with $RGB_d = (1, 1, 1)$ and $A_d = 1$ and use back-to-front compositing, to that we only need to update RGB_d:

$$RGB'_d = (1 - A_s)RGB_d + A_s RGB_s. \tag{1}$$

3.4 Gaussian falloff (3 pts)

3D Gaussian function are of the form

$$g(\mathbf{x}) = e^{-\frac{1}{2}(\mathbf{x}-\mathbf{c})^T \Sigma^{-1}(\mathbf{x}-\mathbf{c})}, \tag{2}$$

where $\mathbf{c} \in \mathbb{R}^3$ is the function's center (location of its maximum), and $\Sigma \in \mathbb{R}^{3\times3}$ is a positive semi-definite covariance matrix that defines its orientation and scaling. This can be easily seen through eigenvector decomposition $\Sigma = R\Lambda R^T = RSS^T R^T$, where R is an orthonormal matrix and therefore a rotation matrix (eigenvectors can be flipped so that $\det(R) = 1$), and S is a diagonal matrix and therefore a scaling matrix.

Update the renderer by varying the alpha channel according to the distance from the center of the splat. Multiplying the alpha channel before alpha blending with $g(\mathbf{x})$. You can use uniform scaling with $\Sigma = s/z$.

3.5 *Optional: non-uniform scaling (2 pts)*

Gaussian functions have two very useful properties that are useful for splatting:

- *they are closed under affine transformations (but not under projective transformations), and*
- *they are closed under integration along a given direction (orthogonal projections).*

You can thus represent a rotated, scaled, and translated splat by just modifying the function's covariance matrix Σ and center $\mathbf{c}$.

You have already implemented translation, now add correct rotation and scaling of the splats. Use the scalars and quaternions from the input file to scale and rotate each splat in 3D, then calculate its 2D covariance matrix in screen space and use it during rendering. You will have to approximate the 2D covariance matrix, as the perspective projection does not preserve the Gaussian function. You can use the approximation method presented in [ZPvBG01], look for the matrix J.

3.6 *Optional: interactivity (2 pt)*

Enable interaction with the camera or the model with the mouse or keyboard. A suitable choice in this case would be an orbit camera. Keep in mind that you must sort the splats every time the camera configuration changes.

3.7 *Optional: fast sorting (2 pts)*

Implement a faster sorting algorithm that enables your implementation to work in real time. If you are working with a GPU, a good choice would be bitonic sort or Batcher's sort (both are examples of sorting networks).

4 Outputs

The expected output of this homework are code for Gaussian splatting, rendered images of the testing data sets, and a report with performance measurements.

References

[KKLD23] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.*, 42(4), jul 2023.

[ZPvBG01] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. EWA volume splatting. In *Proceedings of the Conference on Visualization '01*, VIS '01, page 29–36, USA, 2001. IEEE Computer Society.