CSCI 1430 Final Project Report: 3D Gaussian Splatting

Faisal Zaghloul, Brady Garrison , Arin Idhant, Gurpartap Singh

Team name: Bugs On Windshield Go ... Splat.
TA name: Melvin He

Brown University

Abstract

3D Gaussian splatting is at the forefront of 3D-reconstruction methods with life-like renders of scenes from novel viewpoints. We re-implement a simple trainable 3D-Gaussian model that can recover spatial information from a set of 2D images taken both from real and synthetic scenes. The pipeline involves initializing gaussians, rendering them using an off-the-shelf differentiable gaussian rasterizer, and correcting for rendering errors in an optimization loop.

1. Introduction

Radiance methods are used to turn 2-Dimensional images in 3-Dimensional scenes that could be readily explored using 3-dimensional viewers. These methods use various shapes to build these 3D scenes, and one such method is known as 3D Gaussian splatting. The method iteratively optimizes a set of Gaussians in 3D space using a set of ground-truth images coupled with their respective camera poses. The goal of this project is to optimize the 3D Gaussians which are rendered using image inputs and camera calibration information. Using machine learning, we optimize the 3D-Gaussian-rendered scene through densification to get an output that is close to the ground truth.

For this project we used our pipeline to reconstruct both real-life scenes such as the the B & H 168 classroom, as well as synthetically generated simple scenes (we used Blender) such as a model of Suzanne the monkey.

The overall method consists of a few steps:

1. Generate a point-cloud from the input images to initialize the positions and colors of the Gaussians.
2. Construct a scene that takes the input point-cloud and the original 2D images along with their respective camera poses.
3. Render the scene from the viewpoint of a chosen camera.
4. Compare the resulting image with the original ground-truth.
5. Optimize the original gaussian parameters using gradient descent.

3D Gaussian splatting is a cutting edge technique, the entire team to is happy to have attempted to reimplemnt it, and happy with our results! By the time of our deadline, we were unable to get a full-blown 3D viewer working with our output to properly view our results, and see this task as our next step.

2. Related Work

This project attempts to recreate the ”3D Gaussian Splatting for Real-Time Radiance Field Rendering” which was presented at SIGGRAPH 2023 by Bernhard Kerbl, Georgios Kopanans, Thomas Leimkühler, George Drettakis [1].

Up until this point, nueral radiance fields known as NeRFs[2] were the method of choice for generating high-quality reconstruction, with other point-based algorithms also existing. However, this approach involved long times for training and rendering times. In contrast, Gaussian splatting uses rasterization to render Gaussians, which runs in real-time on most modern systems.

The paper above carefully detailed the entire optimization process which we closely studied before re-implementation as well as provided us with the learning rates and parameters we inevitably found to give us the best results.

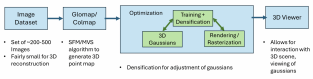


Figure 1. Our Pipeline

Figure 2. Paper's Pipeline

3. Method

Gaussian splatting utilizes gradient descent optimization to fit 3D Gaussians to a scene. A 3D Gaussian is a normal distribution across 3 spatial dimensions. This forms a "blob" in 3D space that can be easily represented, stored, and rendered. The primary driver of this pipeline is the differentiable Gaussian rasterizer[1]. This rasterizer is able to back-propagate gradients back to its inputs, which makes optimization using gradient descent possible.

The first part of this process is to gather information about a scene which includes 3D points and camera calibration for the image inputs. For real-life datasets, we use GLOMAP[3] to perform Structure from Motion (SfM)[4] in order to recover the camera poses and get a sparse point-cloud representation of the scene. This point-cloud is used to bootstrap the initial locations and colors of the Gaussians. We optionally also used Multi-View Stereo (MVS)[5] in order to densify the point-cloud (i.e. generate more points in our point cloud). We did run our optimization workflow with randomly-initialized Gaussians as well, which tends to perform worse than point-cloud-initialized ones (although we haven't carefully evaluated that in our work). The 3D points along with their RGB values are used to create an initial scene generation using a Gaussian rasterizer.

The scene itself is described by the following parameters:

1. Gaussian Parameters (trainable):
   (a) Position in space (aka mean) (x, y, z)
   (b) Scale (aka variance) in all 3 dimensions.
   (c) Rotation as a quaternion
   (d) Opacity
   (e) Color
2. Camera parameters, both intrinsic and extrinsic (non-trainable)

The next step is to optimize our 3D scene to make a life-like reconstruction. This is done through machine learning, where each image, along with it's camera pose is used to determine what changes need to be made to our Gaussians parameters. If more or less Gaussians are needed to represent our scenes, densification occurs where Gaussians are split or cloned as needed.

Optimization of Gaussians in this algorithm uses a gradient descent. Unlike deep-learning methods, there are no traditional layers such as matmuls or convolutions. Instead, the Gaussians parameters themselves are trainable and can be optimized to best fit a scene. There are three aspects that are most important to understanding the inner workings of Gaussian splatting optimization:

- Gaussians with trainable parameters Our representation of Gaussians in a scene have five trainable parameters. These are position, opacity, rotation, and scale. When the optimizer finds gradients for each of these parameters, they are applied to the Gaussians so that they are properly moved and scaled to better fit the scene

- Loss and Differentiable Rasterizer The loss function compares a rendered image of the 3D gaussians to one of the training images that is in the same position. So, on each iteration, the Gaussians are passed through the renderer to compute graidents. The rasterizer itself is able to compute graidents, and passes them back to the Gaussians during the backwards pass. The loss function uses a combination of Structural Similarity Index Measure (SSIM)[6] and L1 loss to determine whether the rendered image is similar to the original.

- Densification After a number of iterations specified by a hyperparameter (in our case, we chose every 5 iterations), the process of densification occurs. Densification involves the removal and addition of Gaussians to better represent the scene. This is distinct from the process of applying gradients to the Gaussians. If a Gaussian is too large to represent a given feature (overconstructed), it is split into two smaller Gaussians that are placed within the bounds of the original Gaussian using a normal distribution. If a Gaussian is too small to represent a given feature (underconstructed), it is cloned into two Gaussians that will later be moved by optimization to represent the scene in greater detail. If the transparency of the gaussian is too high or the Gaussian is tool large to fit on the screen, it will be "pruned" and removed from the representation. Figure 2 created by Kerbl et al [1] shown above visualizes this process.

The learning parameters and hyperparameters that gave us the best results are written in the Tables 1 and 2. We had experimented with different parameters, but found the parameters used by Kerbl and team to be the best in the optimization process.

Learning rates	Value
Opacity	0.05
Scale	0.005
Rotation	0.001
Position	0.0001 Color
0.0025	

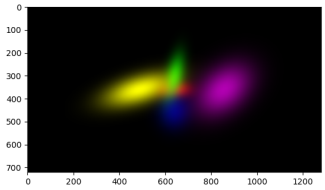
Table 1. The learning rates for each different trainable parameter

Hyperparameters	Value
Regularization Weight	0.01
Densification	2
Opacity Reset Interval	30
Densift Until Epoch	70 DSSIM Scale
0.02	

Table 2. The Hyperparameters that allowed the Gaussians to converge to the best results.

4. Results

Some of our initial outputs simply involved rendering Gaussians to get an intuitive understanding of how they were being rendered by the Gaussian Rasterizer. The following figure showcases five Gaussians rendered by hard coding their position, color, opacities, and rotations.



We then proceeded with representing 2-D images from Gaussians. This process involved a lot of fine-tuning and tweaking the optimization process so that the Gaussians would converge and produce a realistic representation of the target image. Over the outputs that were produced during training, we could see the various densification steps in action. If the Gaussians were under-constructing the image, cloning would take place and we could see new freshly cloned Gaussians. In the subsequent training iterations, we could see these Gaussians optimize to the target image and match it better. By the end of training, we could see fine textures on the trees which were represented by a larger number of Gaussians to capture the high frequency detail of the forest. The Gaussians were most noticeable on the cliffs.

We then proceeded to a 3-D synthetic dataset of Blender's Suzanne. Our synthetic dataset contained renders from Blender at different viewing angles. We then proceeded with the pipeline as we would with a real dataset and obtained our initial point cloud. The Gaussians are rendered using a Python script that revolves around the monkey and saves the render of the Gaussians from that specific camera angle.

Last but not least, we have the Point Cloud reconstruction of our classroom using Glomap made using a dataset that we generated.

4.1. Technical Discussion

An interesting area of discussion is the performance comparison between Gaussian Splatting and its counterpart NeRF (Neural Radiance Fields). This comparison creates an interesting parallel to that of raster and offline rendering within Computer Graphics. While Gaussian Splatting seeks to use rasterization techniques that allow for it to be embarrassingly parallel, NeRFs on the other hand use ray-marching which can be computationally expensive. Efficiency is thus at the heart of Gaussian Splatting. We sought to take this up a notch by using Glomap and Multi-View Stereo to make the process more efficient. Colmap (used by the original paper) offers higher accuracy however it can be computationally expensive. Using Glomap on its own, though efficient, does not produce accurate point clouds. We improve the accuracy by using multi-view stereo to generate denser and more accurate point clouds. It is here that we must consider the trade-offs between efficiency and accuracy. Future developments in the field may choose to focus on one depending on the application but in the real-time context, efficiency is undeniably important.

The paper also presents an interesting challenge of rasterizing Gaussians. Traditional Graphics techniques use triangles as the fundamental building block for 3-D representations. Triangles are well defined primitives that can be rasterized in a relatively straightforward manner. Gaussians on the other hand are distributions in 3-D and traditional discrete approaches towards rasterization will fail on them. The paper thus adopts a differentiable rasterizer that is integrated with the optimization step of the project.

5. Conclusion

Reconstruction, as we know, is one of the fundamental 3 R's of computer vision. Reconstruction of scenes using 2-D images and videos is of immense importance to fields like Medical Imaging, Robotics, Autonomous Vehicles, Computer Graphics and Vision. Further, generating interactive and real-time representations of these 3-D scenes is still an area of active research in the field of Computer Vision. Gaussian Splatting is the state-of-the-art method that has emerged recently that seeks to leverage the efficiency of raster graphics to generate 3-D reconstructions that can be interacted with in realtime. Through our project, we have successfully created 3-D representations of our scenes obtained from 2-D pictures using Gaussians. We re-implemented several parts

Figure 3. The top left image shows the newly cloned Gaussians that will be optimized in subsequent steps. The top center image shows these Gaussians fitting onto the target image. The top right image represents the final composition, which is entirely composed of Gaussians. The middle row shows the reconstruction of Suzanne, while the bottom row displays our classroom reconstruction from Glomap and various Gaussian compositions applied to images from the paper's dataset.

of the state-of-the-art optimization logic and supplemented the original pipeline by introducing Glomap and multi-view stereo as a potential alternative to using Colmap for the SFM component of the project.

References

[1] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. 1, 2

[2] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1

[3] Linfei Pan, Dániel Baráth, Marc Pollefeys, and Johannes Lutz Schönberger. Global structure-from-motion revisited. In *European Conference on Computer Vision (ECCV)*, 2024. 2

[4] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[5] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 2

[6] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 2

Appendix

Team contributions

Brady Garrison I helped to research the optimization loop and construct a framework for our optimization algorithm. I primarily worked on densification after the initial optimization loop had been constructed. I also worked on training and tuning hyper-parameters to ensure that scene representations were as accurate as possible. I also developed the script that allowed us to collect images throughout the training process and convert them into videos. This included rotations of the camera around the scene to view the training process from multiple angles.

Gurpartap Singh The first step was reading the dense paper that outlined what 3D gaussian splatting was, and then had to learn about the graphics side of this project. My main task was working with our datasets. I wrote COLMAP scripts that took in image sets and returned

information including camera intrinsics, height, width, 3d points coords, colors, and information on each images such as translation and quaternions vectors. We also ran MVS on these images and outputs in order to create an even more dense to start with. I then used this information to calculate camera matrices including projection, intrinsic, and extrinsic that we used along with our points to create a 3D scene. I also helped in writing scripts for information necessary for our 3D viewer, but ultimately it did not work.

Faisal Zaghloul Worked on the overall pipeline. I set up the repository with all the dependencies, created some unit tests for the differentiable gaussian rasterizer and the SSIM loss code. I then created the initial dataloader, scene description, camera description, and the end-to-end optimization loop. I also generated the synthetic Blender data, and provided test images to over-fit on to ensure that the pipeline works as expected. Afterwards I fell into a support and integration role, where I helped folks with their various tasks, and integrated all the code together.

Arin Idhant I worked on the SIBR realtime viewer for the project. Initially, I helped setup and drive the CMake build process for the viewer. The viewer had a remote component that could integrate with the optimization step in the original process using a server. The pipeline involved parsing camera intrinsics, generating Camera extrinsics from images, and finally rendering using OpenGL. While I was successful in implementing the network connection between our modified version of the viewer and the training script we implemented, I had some issues initializing the Camera data for the viewer's MultiViewManager. Further I read and prepared docs on the original Gaussian Splatting paper and became familiar with the SIBR viewer's expansive code base.