Pacific Graphics 2025
M. Christie, N. Pietroni, and Y.-S. Wang
(Editors)

COMPUTER GRAPHICS *forum*
Volume 0 (1981), Number 0

Gaussians on their Way: Wasserstein-Constrained 4D Gaussian Splatting with State-Space Modeling

Supplementary Material

1. Numerical Stability in Wasserstein Geometry Computations

In the main paper, we introduced a method for modeling Gaussian dynamics using Wasserstein geometry, involving logarithmic and exponential maps between Gaussian distributions. While the theoretical framework provides a solid foundation, practical implementation requires careful attention to numerical stability to ensure accurate and robust computations. In this supplementary material, we detail the numerical stabilization techniques employed in our algorithms, particularly in the computations of the logarithmic and exponential maps in the Wasserstein space.

1.1. Potential Numerical Issues

Computations involving covariance matrices, such as matrix square roots, inverses, and eigenvalue decompositions, can be numerically unstable in certain scenarios, especially when dealing with matrices that are close to singular or have very small eigenvalues. Potential issues include:

- **Non-positive definiteness**: Covariance matrices must be symmetric positive definite (SPD), but numerical errors can introduce negative eigenvalues.
- **Small or negative eigenvalues**: Eigenvalues close to zero or negative can cause instability in operations like inversion or square roots.
- **Loss of symmetry**: Operations can lead to matrices that are not perfectly symmetric due to floating-point errors.
- **Division by small numbers**: Dividing by very small numbers can amplify numerical errors.

To mitigate these issues, we incorporate several numerical stabilization techniques in our implementation.

1.2. Stabilization Techniques

Our stabilization strategies include:

- **Clamping**: We ensure that variables like scale parameters and eigenvalues are clamped to a minimum positive value (e.g., $\epsilon = 1 \times 10^{-8}$) to prevent division by zero or taking square roots of negative numbers.
- **Symmetrization**: Matrices are enforced to be symmetric by averaging them with their transpose.
- **Adding Identity Matrix**: Small multiples of the identity matrix are added to covariance matrices to ensure positive definiteness.
- **Eigenvalue Clipping**: Eigenvalues are clipped to be non-negative before operations like square roots.
- **Safe Square Roots and Inversions**: Operations are performed in a way that avoids taking square roots or inverses of negative or zero values.

2. Detailed Algorithms with Numerical Stabilizations

We provide detailed algorithms for the logarithmic and exponential maps with explicit numerical stabilization steps.

2.1. Logarithmic Map Computation

Given two Gaussian distributions $\mathcal{N}_t(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ and $\mathcal{N}_{t-1}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$, we compute the velocity $v_t = -\log_{\mathcal{N}_t} \mathcal{N}_{t-1}$. The steps are detailed in Algorithm 1.

2.2. Exponential Map Computation

Using the velocity $v_t = (\Delta\boldsymbol{\mu}_t, \Delta\boldsymbol{\Sigma}_t)$, we predict the next Gaussian distribution $\mathcal{N}^P_{t+1} = \exp_{\mathcal{N}_t}(v_t)$. The steps are detailed in Algorithm 2.

2.3. Matrix Square Root Function

We compute the matrix square root using eigenvalue decomposition with numerical stabilization, as shown in Algorithm 3.

2.4. Sylvester Equation Solver

Solving the Sylvester equation $\Gamma\boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_t\Gamma = \Delta\boldsymbol{\Sigma}_t$ is critical and requires numerical care, as detailed in Algorithm 4.

2.5. Wasserstein Distance Computation

We compute the Wasserstein distance between two Gaussians with numerical stabilization, as shown in Algorithm 5.

Algorithm 1 Numerically Stable Logarithmic Map Computation.

Input: $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}$
Output: $\Delta\boldsymbol{\mu}_t, \Delta\boldsymbol{\Sigma}_t$

1: **Compute Mean Difference:**

$$\Delta\boldsymbol{\mu}_t = \boldsymbol{\mu}_t - \boldsymbol{\mu}_{t-1}$$

2: **Symmetrize Covariance Matrices:**

$$\boldsymbol{\Sigma}_t = \frac{\boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_t^\top}{2}$$

$$\boldsymbol{\Sigma}_{t-1} = \frac{\boldsymbol{\Sigma}_{t-1} + \boldsymbol{\Sigma}_{t-1}^\top}{2}$$

3: **Add Small Identity for Positive Definiteness:**

$$\boldsymbol{\Sigma}_t \leftarrow \boldsymbol{\Sigma}_t + \epsilon\mathbf{I}$$

$$\boldsymbol{\Sigma}_{t-1} \leftarrow \boldsymbol{\Sigma}_{t-1} + \epsilon\mathbf{I}$$

4: **Compute Square Roots:**

$$\boldsymbol{\Sigma}_t^{1/2} = \text{MatrixSqrt}(\boldsymbol{\Sigma}_t)$$

$$\boldsymbol{\Sigma}_t^{-1/2} = (\boldsymbol{\Sigma}_t^{1/2})^{-1}$$

5: **Compute Intermediate Matrix:**

$$\mathbf{P} = \boldsymbol{\Sigma}_t^{1/2}\left(\boldsymbol{\Sigma}_t^{-1/2}\boldsymbol{\Sigma}_{t-1}\boldsymbol{\Sigma}_t^{-1/2}\right)^{1/2}\boldsymbol{\Sigma}_t^{1/2}$$

6: **Symmetrize P and Compute $\Delta\boldsymbol{\Sigma}_t$:**

$$\mathbf{P} = \frac{\mathbf{P} + \mathbf{P}^\top}{2}$$

$$\Delta\boldsymbol{\Sigma}_t = 2\boldsymbol{\Sigma}_t - \mathbf{P} - \mathbf{P}^\top$$

Algorithm 2 Numerically Stable Exponential Map Computation.

Input: $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \Delta\boldsymbol{\mu}_t, \Delta\boldsymbol{\Sigma}_t$
Output: $\boldsymbol{\mu}_{t+1}^P, \boldsymbol{\Sigma}_{t+1}^P$

1: **Predict Mean:**

$$\boldsymbol{\mu}_{t+1}^P = \boldsymbol{\mu}_t + \Delta\boldsymbol{\mu}_t$$

2: **Solve Sylvester Equation for Γ:**

$$\Gamma\boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_t\Gamma = \Delta\boldsymbol{\Sigma}_t$$

3: **Compute Predicted Covariance:**

$$\boldsymbol{\Sigma}_{t+1}^P = \boldsymbol{\Sigma}_t + \Delta\boldsymbol{\Sigma}_t + \Gamma\boldsymbol{\Sigma}_t\Gamma^\top$$

4: **Symmetrize and Ensure Positive Definiteness:**

$$\epsilon \leftarrow 1 \times 10^{-8}$$

$$\boldsymbol{\Sigma}_{t+1}^P = \frac{\boldsymbol{\Sigma}_{t+1}^P + \boldsymbol{\Sigma}_{t+1}^{P\top}}{2} + \epsilon\mathbf{I}$$

Algorithm 3 Matrix Square Root with Stabilization.

Input: Symmetric positive definite matrix $\mathbf{A}$
Output: $\mathbf{A}^{1/2}$

1: **Eigenvalue Decomposition:**

$$\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top$$

2: **Clamp Eigenvalues:**

$$\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$$

$$\epsilon \leftarrow 1 \times 10^{-8}$$

$$\lambda_i \leftarrow \max(\lambda_i, \epsilon) \quad \forall i$$

3: **Compute Square Roots of Eigenvalues:**

$$\boldsymbol{\Lambda}^{1/2} = \text{diag}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \sqrt{\lambda_3})$$

4: **Reconstruct Matrix Square Root:**

$$\mathbf{A}^{1/2} = \mathbf{Q}\boldsymbol{\Lambda}^{1/2}\mathbf{Q}^\top$$

Algorithm 4 Numerically Stable Sylvester Equation Solver.

Input: Matrices $\boldsymbol{\Sigma}_t, \Delta\boldsymbol{\Sigma}_t$
Output: Solution Γ

1: **Vectorize Equation using Kronecker product:**

$$(\boldsymbol{\Sigma}_t \otimes \mathbf{I} + \mathbf{I} \otimes \boldsymbol{\Sigma}_t)\text{vec}(\Gamma) = \text{vec}(\Delta\boldsymbol{\Sigma}_t)$$

2: **Solve for vec(Γ) using linear system solver:**

$$\text{vec}(\Gamma) = (\mathbf{K})^{-1}\text{vec}(\Delta\boldsymbol{\Sigma}_t), \quad \text{where } \mathbf{K} = (\boldsymbol{\Sigma}_t \otimes \mathbf{I} + \mathbf{I} \otimes \boldsymbol{\Sigma}_t)$$

3: **Reshape vec(Γ) to Matrix Form:**

$$\Gamma = \text{reshape}(\text{vec}(\Gamma))$$

3. Additional Implementation Details

3.1. Clamping and Threshold Values

We use small positive values (e.g., $\epsilon = 1 \times 10^{-8}$) to prevent numerical instabilities such as division by zero or square roots of negative numbers. These values are chosen empirically to balance stability and computational accuracy.

3.2. Symmetry Enforcement

To ensure matrices remain symmetric after operations that may introduce asymmetry due to floating-point errors, we symmetrize matrices by averaging them with their transpose.

3.3. Positive Definiteness of Covariance Matrices

Covariance matrices are enforced to be positive definite by adding a small multiple of the identity matrix. This step is crucial before performing operations like eigenvalue decomposition or matrix inversion.

Algorithm 5 Numerically Stable Wasserstein Distance Computation.

Input: Gaussian parameters $(\boldsymbol{\mu}_1, \mathbf{S}_1, \mathbf{R}_1)$, $(\boldsymbol{\mu}_2, \mathbf{S}_2, \mathbf{R}_2)$
Output: Wasserstein distance W_2

1: **Define Epsilon for Clamping:**

$$\epsilon \leftarrow 1 \times 10^{-8}$$

2: **Clamp Diagonal Elements of Scale Matrices:**

$$\text{diag}(\mathbf{S}_1) \leftarrow \max(\text{diag}(\mathbf{S}_1), \epsilon)$$

$$\text{diag}(\mathbf{S}_2) \leftarrow \max(\text{diag}(\mathbf{S}_2), \epsilon)$$

3: **Compute Covariance Matrices:**

$$\boldsymbol{\Sigma}_1 = \mathbf{R}_1 \mathbf{S}_1 \mathbf{S}_1^\top \mathbf{R}_1^\top$$

$$\boldsymbol{\Sigma}_2 = \mathbf{R}_2 \mathbf{S}_2 \mathbf{S}_2^\top \mathbf{R}_2^\top$$

4: **Compute Squared Euclidean Distance of Means:**

$$D_\mu^2 = \|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|^2$$

5: **Compute Intermediate Matrix for Trace Term:**

$$\mathbf{E}_{12} = \mathbf{R}_1^\top \mathbf{R}_2 \mathbf{S}_2 \mathbf{S}_2^\top \mathbf{R}_2^\top \mathbf{R}_1$$

$$\mathbf{C} = \mathbf{S}_1^{1/2} \mathbf{E}_{12} \mathbf{S}_1^{1/2}$$

6: **Symmetrize C and Compute its Eigenvalues:**

$$\mathbf{C} = \frac{\mathbf{C} + \mathbf{C}^\top}{2}$$

$$\lambda_i \leftarrow \text{Eigenvalues}(\mathbf{C}) \quad \forall i$$

7: **Clamp Eigenvalues and Compute Trace of $\mathbf{C}^{1/2}$:**

$$\lambda_i \leftarrow \max(\lambda_i, \epsilon) \quad \forall i$$

$$\text{Tr}\left(\mathbf{C}^{1/2}\right) = \sum_i \sqrt{\lambda_i}$$

8: **Compute Squared Wasserstein Distance:**

$$W_2^2 = D_\mu^2 + \text{Tr}(\boldsymbol{\Sigma}_1) + \text{Tr}(\boldsymbol{\Sigma}_2) - 2\,\text{Tr}\left(\mathbf{C}^{1/2}\right)$$

9: **Clamp to Non-negative and Take Square Root:**

$$W_2^2 \leftarrow \max(W_2^2, 0)$$

$$W_2 = \sqrt{W_2^2}$$

3.4. Eigenvalue Clamping

When computing eigenvalues for operations like matrix square roots or logarithms, we clamp eigenvalues to a minimum positive value to avoid undefined operations.

3.5. Matrix Inversion and Square Roots

All matrix inversions and square roots are performed with checks to ensure numerical stability. Inversions are only performed on matrices that are well-conditioned, and square roots are taken only of non-negative eigenvalues.

4. Per-Scene Results

We provide detailed per-scene quantitative comparisons on the D-NeRF dataset to demonstrate the effectiveness of our method across various dynamic scenes. Table 1 presents the results for each scene, comparing our method with several state-of-the-art approaches. We provide video demonstrations in the supplementary material, which are rendered from fixed camera viewpoints using interpolated continuous timestamps.

Table 1: *Quantitative comparison on D-NeRF dataset across different scenes. For all metrics, PSNR↑, SSIM↑ indicate higher is better, while LPIPS↓ indicates lower is better. Red, orange and yellow indicate the best, second-best and third-best results respectively.*

Method	Hell Warrior			Mutant			Hook			Bouncing Balls		
	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
DyNeRF [LSZ*22]	24.58	0.9240	0.08070	29.31	0.9472	0.04920	28.02	0.9395	0.06460	29.49	0.9490	0.05230
StreamRF [LSW*22]	23.16	0.9150	0.09100	28.31	0.9372	0.05920	27.02	0.9295	0.07460	28.16	0.9350	0.06230
HyperReel [AHR*23]	25.36	0.9354	0.07350	30.11	0.9551	0.04010	29.63	0.9433	0.05360	30.36	0.9520	0.04230
NeRFPlayer [SCL*23]	27.69	0.9283	0.06100	29.69	0.9451	0.05100	28.69	0.9383	0.06100	30.69	0.9483	0.05100
K-Planes [FKMW*23]	26.58	0.9520	0.08240	32.50	0.9713	0.03620	28.12	0.9489	0.06620	40.05	0.9934	0.03220
4D-GS [YYPZ24]	40.02	0.9155	0.10560	37.53	0.9336	0.05800	32.71	0.8876	0.10340	40.62	0.9591	0.06000
Def-3D-Gauss [YGZ*24]	41.54	0.9873	0.02340	38.10	0.9951	0.00520	42.06	0.9867	0.01440	41.01	0.9953	0.00930
4D-Rotor-Gauss [DWD*24]	39.46	0.9730	0.03300	39.26	0.9670	0.03800	39.89	0.9640	0.02500	39.25	0.9780	0.03100
SC-GS [HSY*24]	41.38	0.9850	0.02899	39.92	0.9904	0.01396	42.83	0.9900	0.01216	41.56	0.9839	0.02081
Ours	41.40	0.9863	0.02960	40.77	0.9941	0.00480	41.52	0.9857	0.01540	42.79	0.9943	0.01030

Method	Lego			T-Rex			Stand Up			Jumping Jacks		
	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
DyNeRF [LSZ*22]	28.58	0.9384	0.06070	29.57	0.9439	0.04870	29.59	0.9401	0.05850	29.61	0.9397	0.05280
StreamRF [LSW*22]	27.16	0.9284	0.07070	28.16	0.9339	0.05870	28.16	0.9301	0.06850	28.16	0.9297	0.06280
HyperReel [AHR*23]	29.36	0.9484	0.05070	30.36	0.9539	0.03870	30.36	0.9501	0.04850	30.36	0.9497	0.04280
NeRFPlayer [SCL*23]	29.69	0.9383	0.06100	30.69	0.9451	0.05100	30.69	0.9483	0.05100	30.69	0.9479	0.05100
K-Planes [FKMW*23]	33.10	0.9695	0.03310	33.43	0.9737	0.03430	33.09	0.9793	0.03100	31.11	0.9708	0.04020
4D-GS [YYPZ24]	31.10	0.9384	0.06070	35.29	0.9619	0.04870	38.11	0.9301	0.07850	34.66	0.9708	0.04080
Def-3D-Gauss [YGZ*24]	33.07	0.9794	0.02050	43.49	0.9933	0.00980	43.24	0.9831	0.00870	37.72	0.9897	0.01550
4D-Rotor-Gauss [DWD*24]	32.24	0.9710	0.03300	38.26	0.9895	0.04160	38.89	0.9620	0.04800	36.61	0.9638	0.04510
SC-GS [HSY*24]	34.35	0.9353	0.05159	42.11	0.9898	0.01351	42.86	0.9944	0.00482	36.35	0.9838	0.00949
Ours	34.74	0.9784	0.01930	43.66	0.9940	0.00880	42.64	0.9945	0.00720	37.91	0.9887	0.01360