Texture-GS: Disentangling the Geometry and Texture for 3D Gaussian Splatting Editing

Fernando
Reviewer

Diana Aldana
Archaeologist

Nanci
Hacker

Victor Ferrari
PhD Student

Institute of Pure and Applied Mathematics
2024

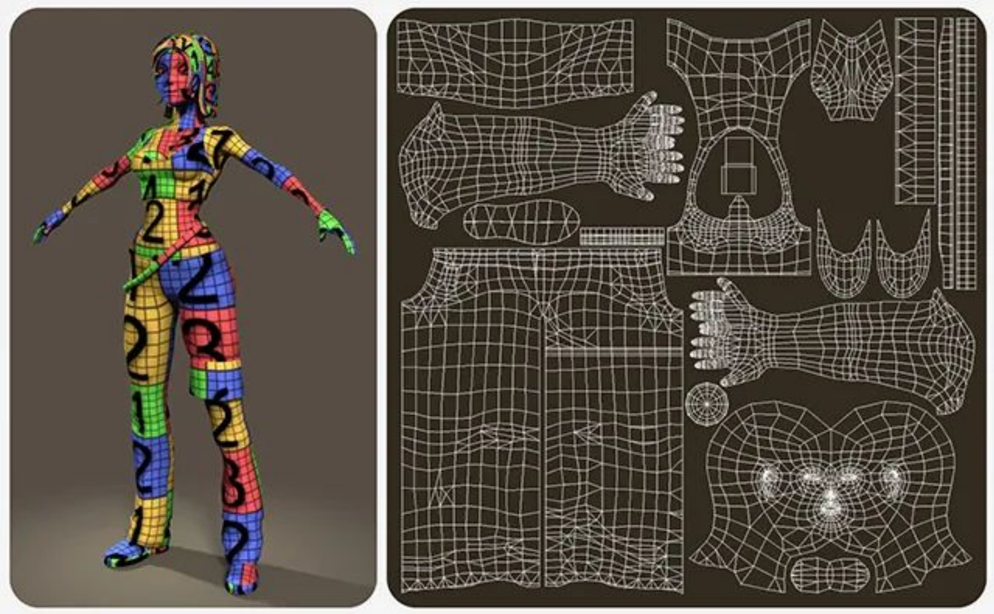
75 anos impa
Instituto de Matemática Pura e Aplicada

Reviewer

Reviewer

Summary:

- **Texture-GS** falls within the range of computer vision techniques aimed at the reconstruction, editing, and real-time rendering of scenes in different applications;



  - Mapping the 3D representation of the scene into 2D UV coordinates using Multilayer Perceptron.
  - The **texture mapping module** incorporates an MLP network and Taylor series to smooth the texture map and its continuity;

Reviewer

Summary:

Remembering, 3D-GS represents the scene as a set of **N** 3D Gaussians $\mathcal{G} = \{G_i(x)\}_{i=1}^{N}$

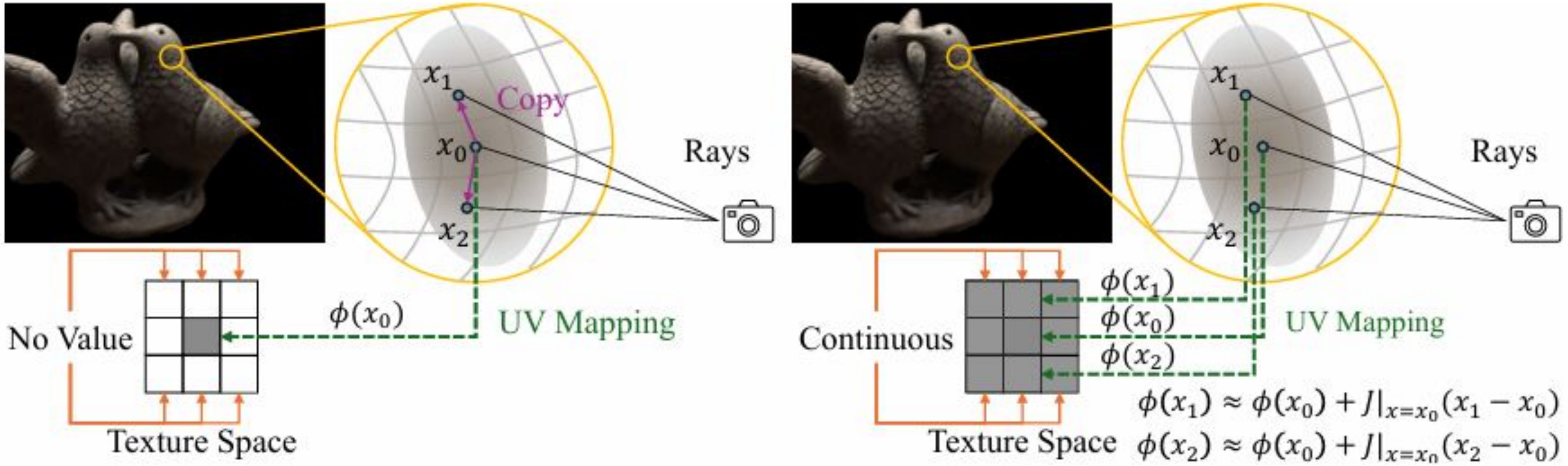
$$G_i(x) = \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma_i^{-1}(x-\mu)\right).$$

The final color at pixel *p* is obtained from a cumulative volumetric rendering of Gaussians, considering transparency alpha

$$C_p = \sum_{j \in \mathcal{N}_p} c_j \alpha_j \prod_{k=1}^{j-1}(1-\alpha_k)$$

Reviewer

Summary:



Reviewer

Strengths:

- **Texture-GS** introduces an explicit and independent method to disentangle the geometry and texture for 3D-GS in an efficient manner
- The novel method inherits the strengths of 3D-GS (a NeRF based method) regarding the use of Gaussians in the scene representation
- Experiments demonstrate the successfully processing for view synthesis, texture swapping and editing with real-time rendering on consumer-level devices

Table 1: Comparison of novel view synthesis results on the DTU dataset.

(a) Comparison with the SOTAs

Method	DTU			
	PSNR↑	L1↓	LPIPS↓	FPS
NeuTex	30.39	0.0158	0.1613	0.025
NGF	29.44	0.0166	0.1506	0.025
3DGS	30.99	0.0121	0.1079	198
Ours	30.03	0.0135	0.1440	58

(b) Different number of 3D Gaussians

#Gauss	DTU			
	PSNR↑	L1↓	LPIPS↓	FPS
100%	30.03	0.0135	0.1440	58
50%	29.57	0.0142	0.1555	69
20%	28.75	0.0155	0.1705	82
5%	27.86	0.0172	0.1841	104

Reviewer

Weaknesses:

- **Text:** Absence of more 3D-GS parameters
    - Discussion about Gaussian initialization
    - Ordering of topics...

Reviewer

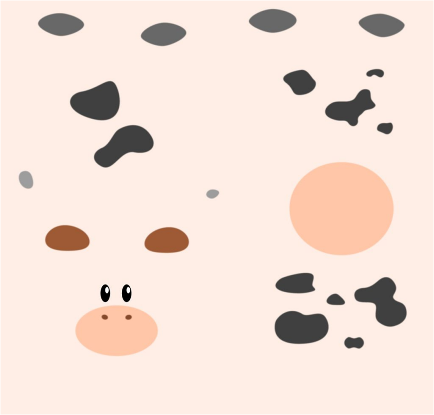
”
“

Rating and Justification:

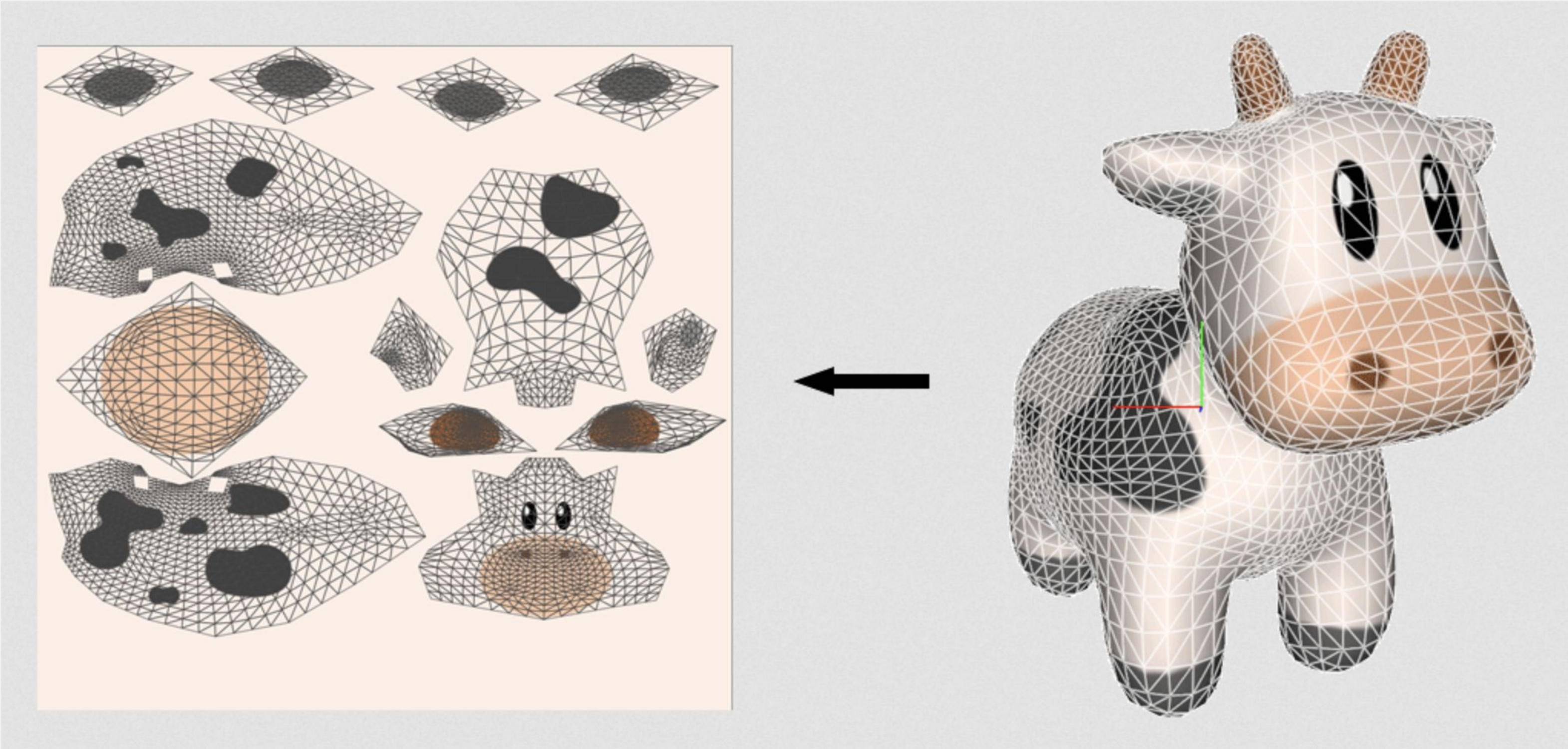
Archaeologist

Context

Classic mesh representation



texture



Context

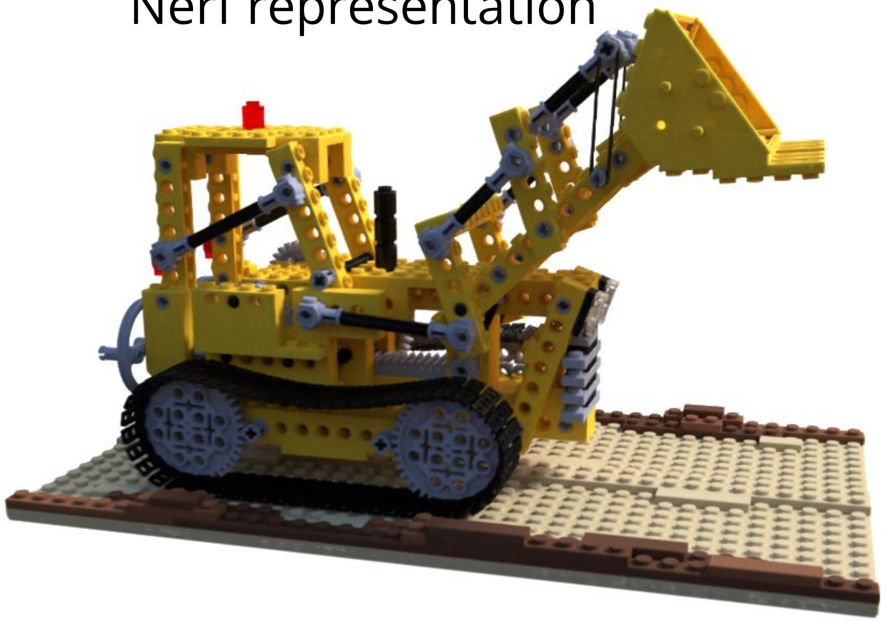
Spot with texture

Spot with other texture

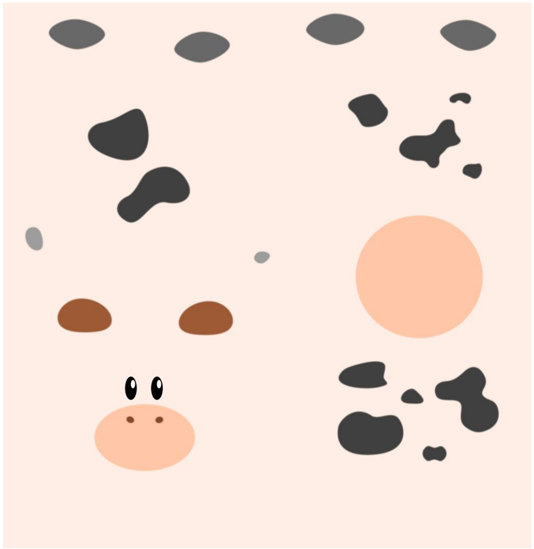
Sphere with spot’s texture

Context

Nerf representation



texture

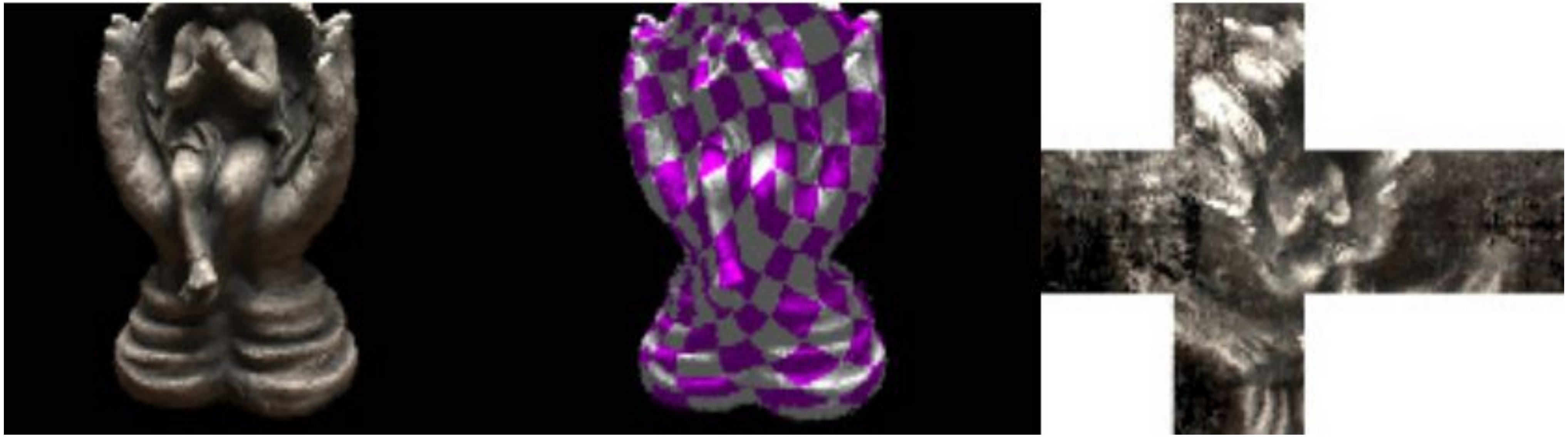


?

Previous papers

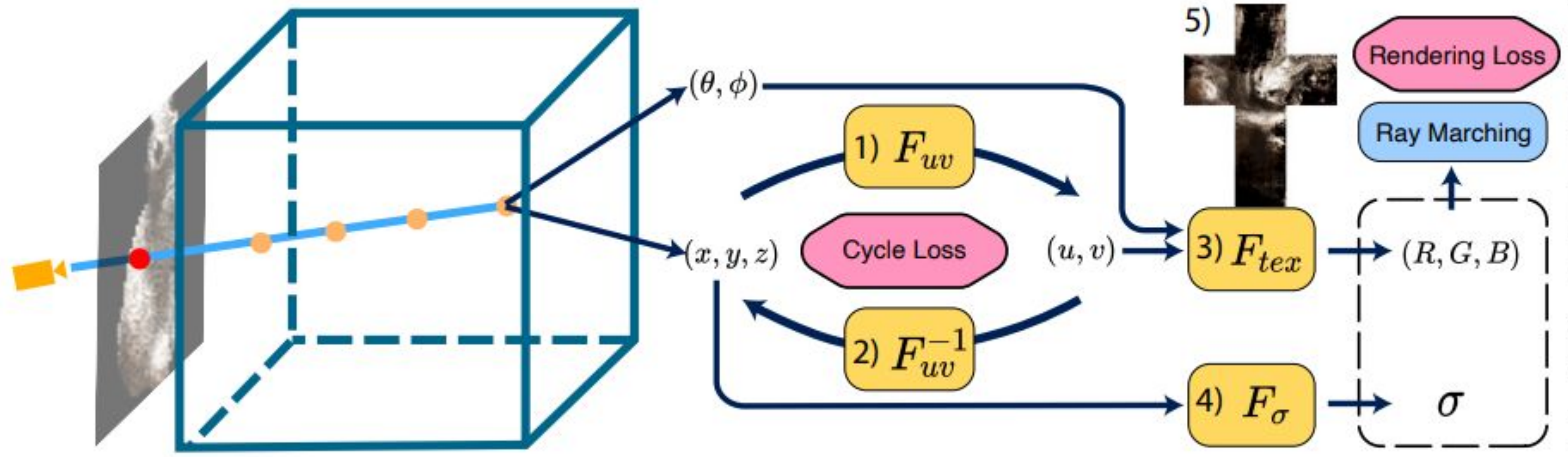
Neutex: Neural texture mapping for volumetric neural rendering. XIANG, Fanbo, et al. (CVPR 2021)

NeRF's geometry decoupling from texture



Previous papers

Neutex: Neural texture mapping for volumetric neural rendering. XIANG, Fanbo, et al. (CVPR 2021)



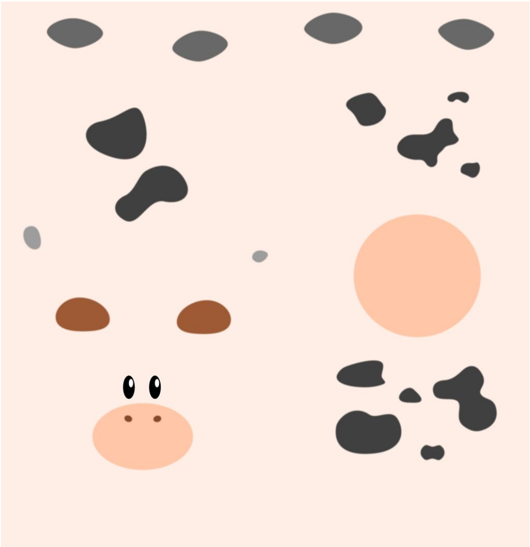
It trains three networks:

- F_σ : Learns geometry.
- F_{uv} : Learns the 3D-to-2D (semi) bijective parameterization.
- F_{tex}: Learns the view dependant radiance.

Relationship with Texture-GS

GS representation

texture

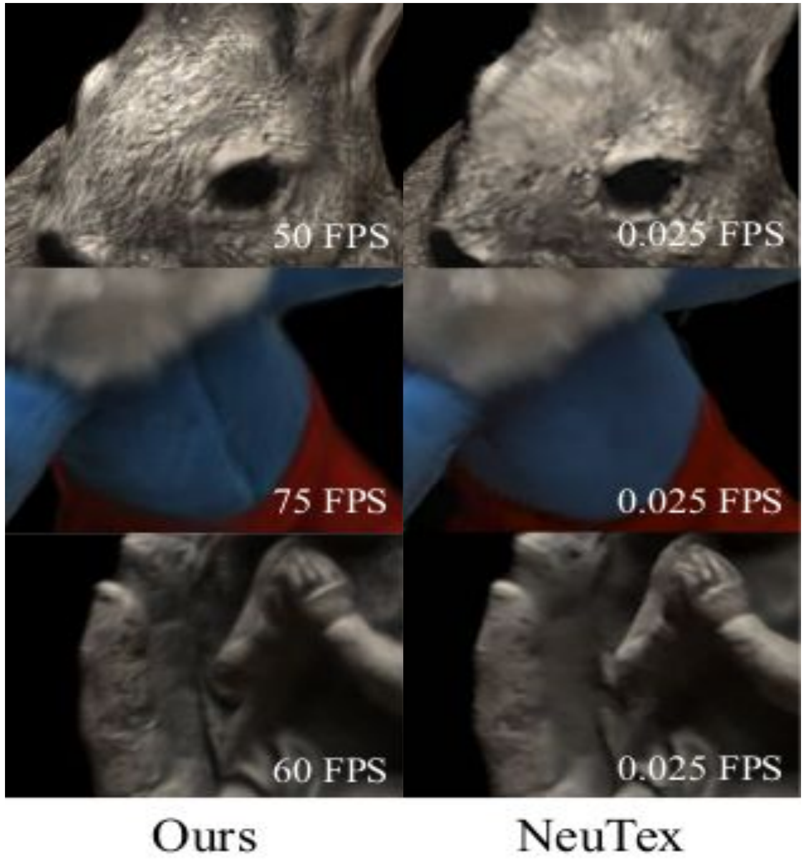


Texture-GS

Relationship with Texture-GS

- Texture-GS is an adaptation of NeuTex method to Gaussian Splatting
- It derives some of their loss terms from NeuTex
  - $L_{\text{cycle}} = \sum_i w_i \| F_{\text{uv}}^{-1}(F_{\text{uv}}(\mathbf{x}_i)) - \mathbf{x}_i \|_2^2.$
  - $L_{\text{mask}} = \| M_{\text{gt}} - (1 - T_N) \|_2^2.$

Method	DTU			
	PSNR↑	L1↓	LPIPS↓	FPS
NeuTex	30.39	0.0158	0.1613	0.025
Ours	30.03	0.0135	0.1440	58

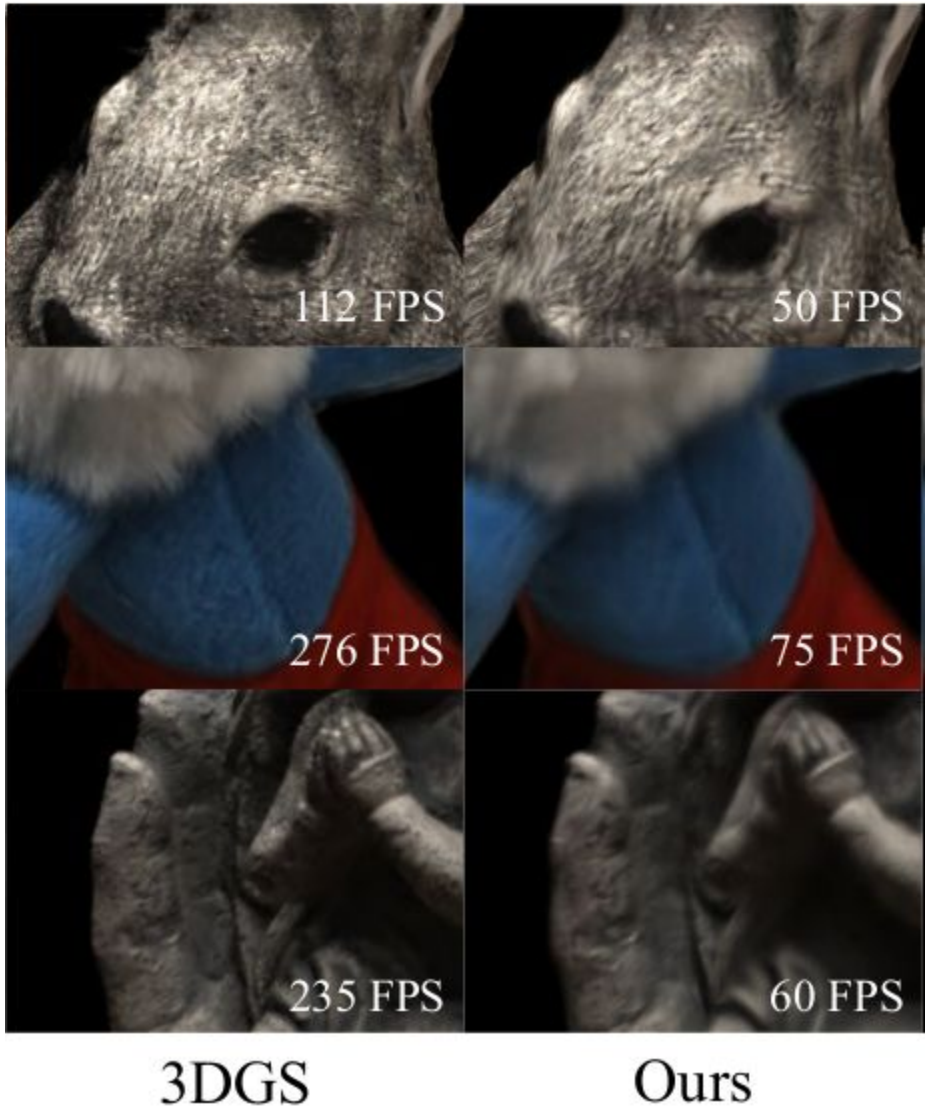


Relationship with Texture-GS

It successfully decouples appearance from geometry.

But it worsens performance from 3D-GS!

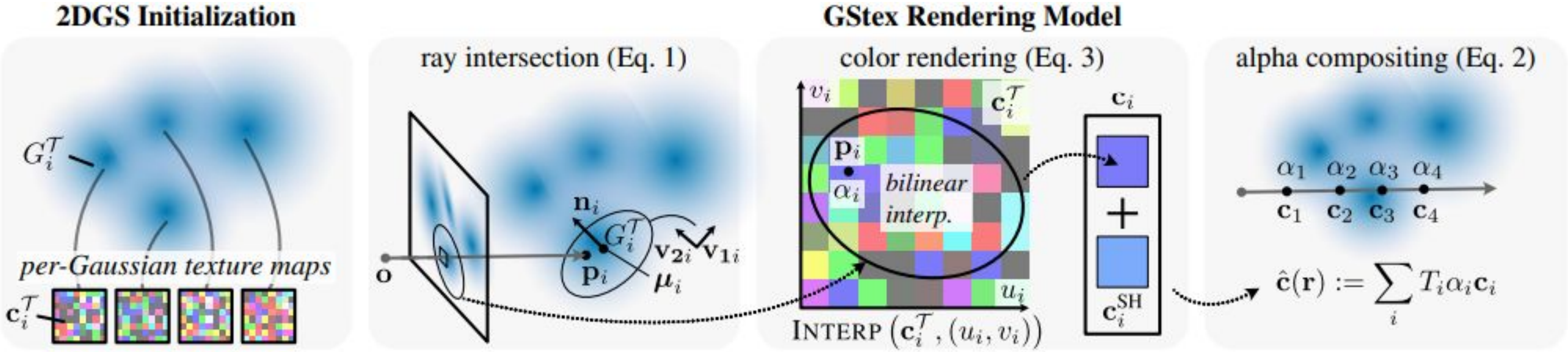
Method	DTU			
	PSNR↑	L1↓	LPIPS↓	FPS
3DGS	30.99	0.0121	0.1079	198
Ours	30.03	0.0135	0.1440	58



GStex: Per-Primitive Texturing of 2D Gaussian Splatting for Decoupled Appearance and Geometry Modeling. Rong, et al.

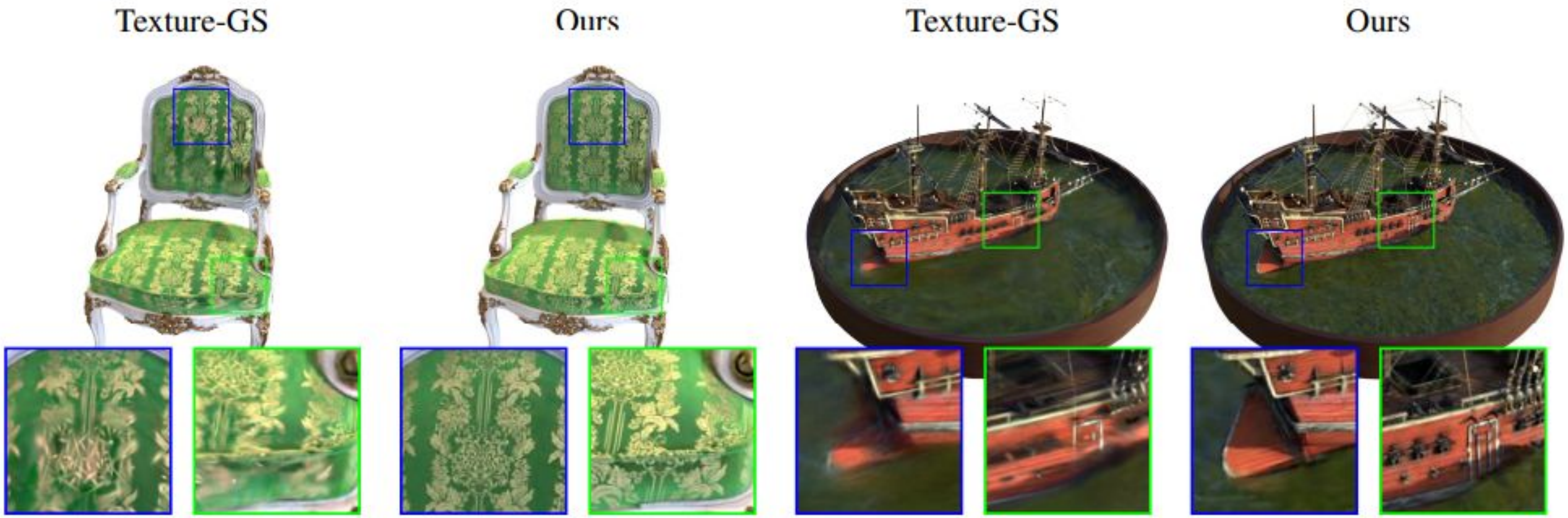
Gaussians are defined by parameters $G^{\mathcal{T}} = \{\mathbf{c}^{\mathcal{T}}, \mathbf{c}^{\text{SH}}, \alpha, \mathbf{R}, \boldsymbol{\mu}, \boldsymbol{\sigma}\}$

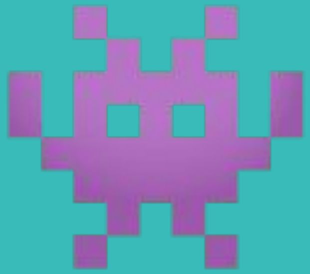
where $\mathbf{c}_i(\mathbf{p}_i, \mathbf{d}) = \underbrace{\text{INTERP}\left(\mathbf{c}_i^{\mathcal{T}}, (u_i(\mathbf{p}), v_i(\mathbf{p}))\right)}_{\text{Diffuse term}} + \underbrace{\text{SH}\left(\mathbf{c}_i^{\text{SH}}, \mathbf{d}\right)}_{\text{View-dependant term}}$



Posterior work

GStex: Per-Primitive Texturing of 2D Gaussian Splatting for Decoupled Appearance and Geometry Modeling. Rong, et al.





HACKER

UV mapping Loss

$$\mathcal{L}_{\mathrm{UV}} = \mathcal{L}_{\mathrm{cycle}}^{\mathrm{3d}} + \mathcal{L}_{\mathrm{CD}} + \mathcal{L}_{\mathrm{cycle}}^{\mathrm{2d}} \qquad (7)$$

Source: https://github.com/slothfulxtx/Texture-GS/blob/main/models/uv_map_gaussian3d.py#L167

UV mapping: 3D cycle consistency loss

$$\mathcal{L}_{\text{cycle}}^{\text{3d}} = \frac{1}{N_d} \sum_{i=1}^{N_d} ||x_i - \phi^{-1} \circ \phi(x_i)|| \quad (4)$$

```
loss = 0.0

uv = self.uv_net(world_xyz, geo_emb)
if loss_cfg.lambda_inverse and self.in_range(cur_iter, loss_cfg.inverse_range):
    world_xyz_inv = self.inv_uv_net(uv, geo_emb)
    Linv = ((world_xyz - world_xyz_inv) ** 2).sum(-1)
    Linv = Linv.mean()
    loss += loss_cfg.lambda_inverse * Linv
    loss_stats.update(Linv=Linv)
```

Source: https://github.com/slothfulxtx/Texture-GS/blob/main/models/uv_map_gaussian3d.py#L186

UV mapping: Chamfer distance loss

$$\mathcal{L}_{\text{CD}} = \frac{1}{N_u}\sum_{i=1}^{N_u}\min_{p_j\in\mathcal{P}}||\phi^{-1}(u_i) - p_j|| + \frac{1}{N_p}\sum_{j=1}^{N_p}\min_{u_i\in\mathcal{U}}||\phi^{-1}(u_i) - p_j|| \tag{5}$$

```
# from pytorch3d.loss import chamfer_distance
...
sample_uvs, sample_inv_xyzs = None, None

if loss_cfg.lambda_chamfer and self.in_range(cur_iter, loss_cfg.chamfer_range):
    if sample_uvs is None:
        sample_uvs = self.inv_uv_net.sample(device=depth.device)
    if sample_inv_xyzs is None:
        sample_inv_xyzs = self.inv_uv_net(sample_uvs, geo_emb)

    Lchamfer, _ = chamfer_distance(sample_inv_xyzs.unsqueeze(0), self.pcd.unsqueeze(0))
    # npts,
    loss += loss_cfg.lambda_chamfer * Lchamfer
    loss_stats.update(Lchamfer=Lchamfer)
```

Source: https://github.com/slothfulxtx/Texture-GS/blob/main/models/uv_map_gaussian3d.py#L197

UV mapping: 2D cycle consistency loss

$$\mathcal{L}_{\text{cycle}}^{\text{2d}} = \frac{1}{N_u} \sum_{i=1}^{N_u} ||u_i - \phi \circ \phi^{-1}(u_i)|| \quad (6)$$

```python
if loss_cfg.lambda_inverse2 and self.in_range(cur_iter, loss_cfg.inverse_range2):
    if sample_uvs is None:
        sample_uvs = self.inv_uv_net.sample(depth.device)
    if sample_inv_xyzs is None:
        sample_inv_xyzs = self.inv_uv_net(sample_uvs, geo_emb)
    sample_inv_uvs = self.uv_net(sample_inv_xyzs, geo_emb)
    Linv = ((sample_inv_uvs - sample_uvs) ** 2).sum(-1)
    Linv = Linv.mean()
    loss += loss_cfg.lambda_inverse2 * Linv
    loss_stats.update(Linv2=Linv)
```

Source: https://github.com/slothfulxtx/Texture-GS/blob/main/models/uv_map_gaussian3d.py#L221

Approximate uv coordinates: Jacobian

$$\tilde{\phi}(I(G_j, r_p)) = \phi(\mu_j) + J|_{x=\mu_j}(I(G_j, r_p) - \mu_j) \quad (14)$$

```python
@property
def get_grad_uvs(self):
    if self._grad_uv is not None:
        return self._grad_uv
    xyz = self._xyz.detach()
    geo_emb = self.geo_emb(torch.zeros(1, dtype=torch.long, device=xyz.device)).squeeze()
    geo_emb = geo_emb.detach()
    def func(inputs):
        return self.uv_net(inputs, geo_emb).float().contiguous().sum(dim=0)
    grad_uvs = jacobian(func=func, inputs=xyz)
    # 3, npts, 3
    return grad_uvs.permute(1, 0, 2).reshape(-1, 9).contiguous().detach().requires_grad_(False)
```

Source: https://github.com/slothfulxtx/Texture-GS/blob/main/models/texture_gaussian3d.py#L217

Approximate uv coordinates: Evaluation

$$\tilde{\phi}(I(G_j, r_p)) = \phi(\mu_j) + J|_{x=\mu_j}(I(G_j, r_p) - \mu_j) \quad (14)$$

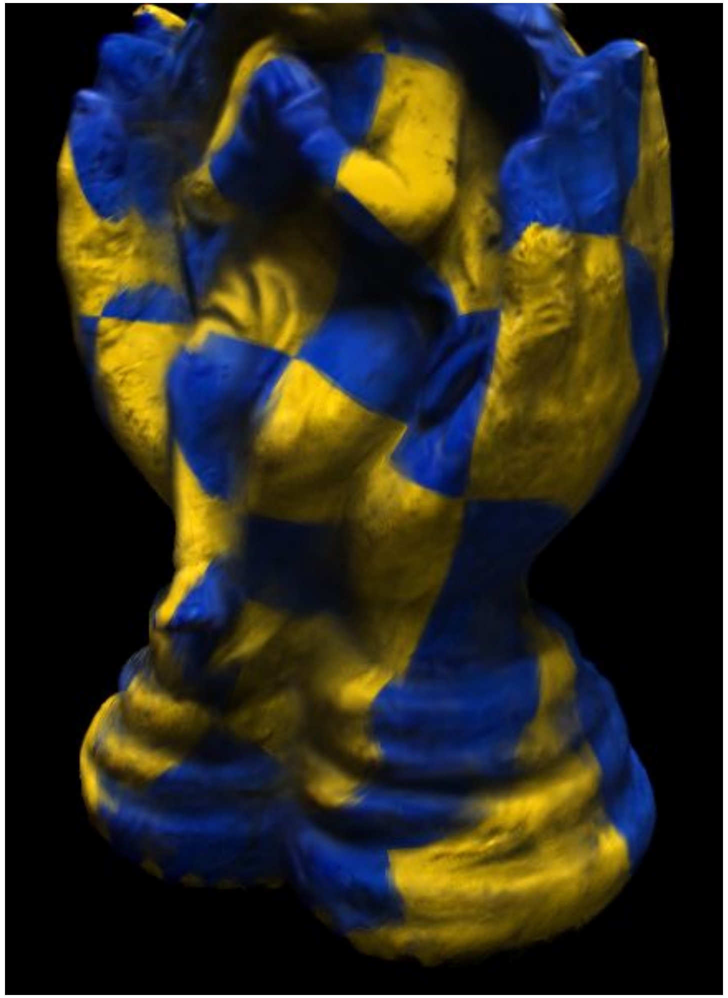
```
float3 orig_point = {orig_points[g_idx * 3], orig_points[g_idx * 3+1], orig_points[g_idx * 3+2]};
float3 norm = {norms[g_idx * 3], norms[g_idx * 3 + 1], norms[g_idx * 3 + 2]};
// (cam_p - orig_point) * norm
float bias = (cam_p.x - orig_point.x)*norm.x + (cam_p.y - orig_point.y)*norm.y + (cam_p.z - orig_point.z)*norm.z;
float denom = pix_dir.x * norm.x + pix_dir.y * norm.y + pix_dir.z * norm.z;
float t;
float3 delta_xyz;
float clamp_radius = clamp_radii[g_idx], delta_norm;
if(fabs(denom) > 1e-6) {
  t = -bias / denom;
  delta_xyz = {cam_p.x + t*pix_dir.x - orig_point.x, cam_p.y + t*pix_dir.y  - orig_point.y, cam_p.z + t*pix_dir.z - orig_point.z};
  delta_norm = sqrt(delta_xyz.x * delta_xyz.x + delta_xyz.y * delta_xyz.y + delta_xyz.z * delta_xyz.z);
  if(delta_norm > clamp_radius)
    delta_xyz = make_float3(delta_xyz.x / delta_norm * clamp_radius, delta_xyz.y / delta_norm * clamp_radius, delta_xyz.z / delta_norm * clamp_radius);
}else{
  delta_xyz = {0, 0, 0};
}
float3 delta_uv = Vec3x3(gradient_uvs + g_idx*9, delta_xyz);
float3 uv = {uvs[g_idx*3] + delta_uv.x, uvs[g_idx*3+1] + delta_uv.y, uvs[g_idx*3+2] + delta_uv.z};
denom = sqrt(uv.x * uv.x + uv.y * uv.y + uv.z * uv.z);
if(denom < 1e-6)
  uv = {uvs[g_idx*3], uvs[g_idx*3+1], uvs[g_idx*3+2]};
else
  uv = {uv.x / denom, uv.y / denom, uv.z / denom};

float3 color = cube_texture_fetch(uv, texture, TR, rgbs[g_idx]);
```

Source: https://github.com/slothfulxtx/diff-gauss-uv-tex/blob/main/cuda_rasterizer/forward.cu#L406

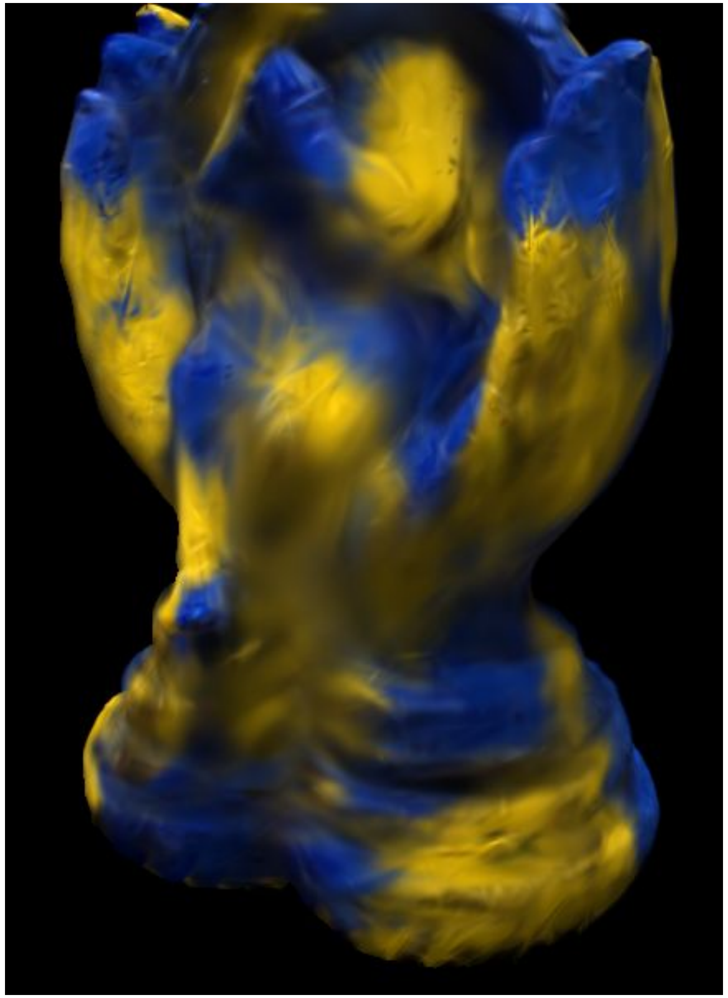
Experiment

Result

Texture-GS



Zero Jacobian



Result: original texture

Texture-GS

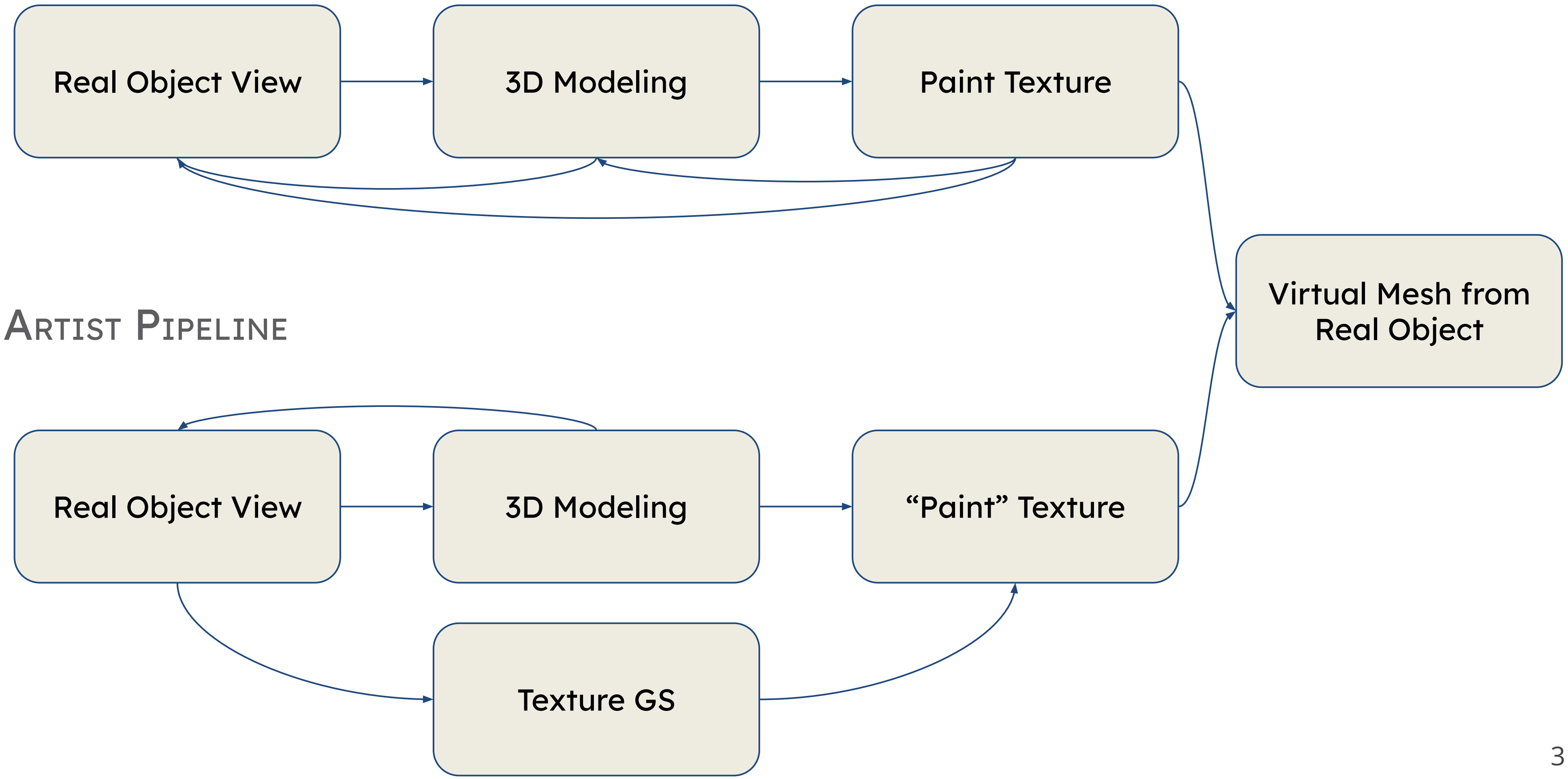
Zero Jacobian

PhD Student

Proposta 1 - "Consumer-level evaluation"

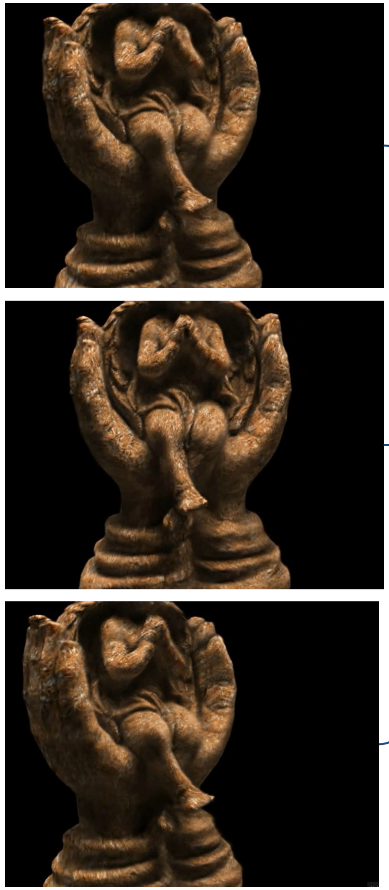
Normal 3D Artist Pipeline

Proposed 3D Artist Pipeline

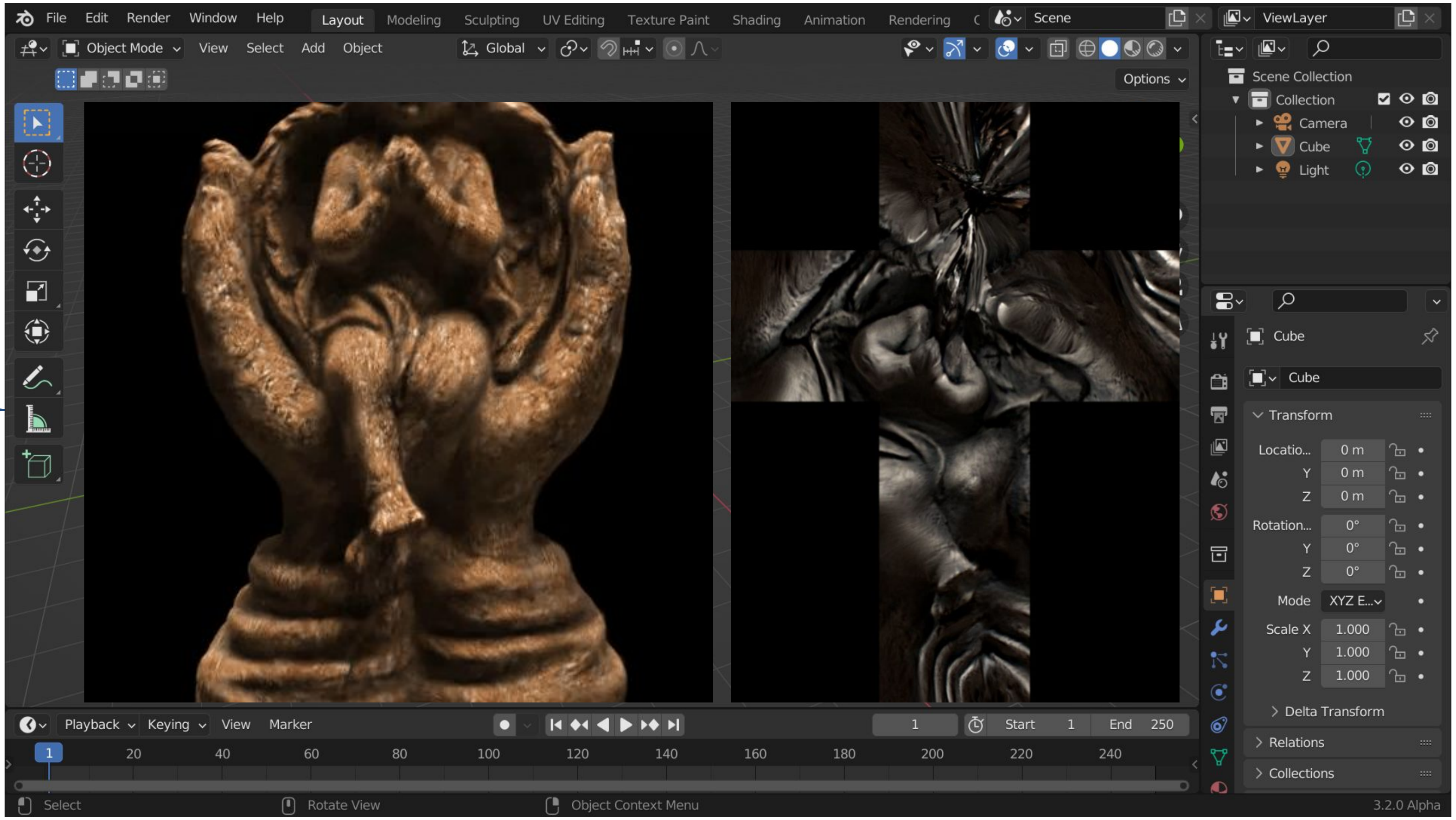


Proposta 1 - "Consumer-level evaluation"

Proposed 3D Artist Pipeline



Views as Inputs



Application

Proposta 2 - "A texture per feature"

Proposta 2 - "A texture per feature"

Thank You!