Context and Techniques for Hybrid Relightable 3D Gaussian Rendering

Table of Contents

Necessary Reads **4**
3D Gaussian Splatting Notes **4**
Introduction 4
Related Work 4
Overview 4
Differentiable 3D Gaussian Splatting 5
Optimization with Adaptive Density Control of 3D Gaussian 5
Fast Differentiable Rasterizer for Gaussians 5
Implementation, Results, and Evaluation 5
Discussion and Conclusions 5
Relightable 3D Gaussian Notes **6**
Introduction 6
Relightable 3D Gaussians 6
Point-based Ray Tracing 7
Experiments 7
3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes Notes **8**
Introduction 8
Related Work 8
Background 8
Method 8
Experiments and Ablations 9
Applications 9
Traditional 3D Polygon Models **10**
Physically Based Rendering (PBR) **11**
The Rendering Equation 11
Bidirectional Reflectance Distribution Function (BRDF) 12
Fresnel Function 13
Diffuse Function 13
Specular Function 14
Normal Distribution Function 14
Geometry Shadowing Function 14
Metallic Rendering Special Case 15
Expanded PBR Rendering Equation 15
Raytracing **15**
Bounding Volume Hierarchy (BVH) Construction 16
Primary Ray Generation 16
Ray-Box Intersection 17
Ray-Triangle Intersection 18
Ray-Gaussian Intersection 19

Glossary.....**20**
Resources.....**21**

Necessary Reads

Please read these papers; they are fundamental to what we are trying to accomplish.

3D Gaussian Splatting for Real-Time Radiance Field Rendering: this original 3D Gaussian paper introduced this novel rendering technique. Read this paper to gain a foundational understanding of how they generate 3D Gaussian models. The rasterization rendering technique should be ignored as we are using raytracing.

Relightable 3D Gaussian: Real-time Point Cloud Relighting with BRDF Decomposition and Ray Tracing: this paper is *basically* what our entire project is based on. They offer a way to decode Gaussian albedo into PBR material properties and a Gaussian raytracing algorithm.

3D Gaussian Ray Tracing of Particle Scenes: this NEW paper (published Oct. 10) expands on the ideas of 3DGS by contributing a unique, optimized GPU-accelerated ray tracing algorithm for semi-transparent particles.

3D Gaussian Splatting Notes

Introduction

The current problem with Neural Radiance Fields (NeRFs), the present technique for high-quality novel-view synthesis, is its long training and rendering times. 3D Gaussians aim to solve these problems. They are initialized from sparse point clouds produced from Structure from Motion (SfM), which are then optimized by tuning Gaussian parameters (3D position, opacity (α), anisotropic covariance, and spherical harmonics (SH) coefficients) and density. 3D Gaussian's are incredibly fast to rasterize.

Related Work

SfM enabled novel view synthesis from a series of photos by generating a point cloud. Multi-view stereo (MVS) uses the geometry to guide the reprojection and blends the input images to create a full 3D reconstruction. Deep learning techniques use Convolution Neural Networks (CNNs) to estimate blend weights. NeRFs and importance sampling of volumetric ray-marching improved quality but resulted in high computational costs. Research has focused on incremental speed improvements using spatial data structures, regularization, and smaller Multi-Layer Perceptions (MLPs). 3D Gaussians are used as opposed to traditional point-based rendering to avoid issues of aliasing and holes.

Overview

Generate a point cloud with SfM from a set of images of a static scene. A set of 3D Gaussians is created for each point in the cloud containing a position, covariance matrix, and opacity.

Gaussian parameters and density are tuned to better fit the image. Rasterizer does alpha-blending of ordered splats.

Differentiable 3D Gaussian Splatting

3D Gaussians are chosen as the primitive for easy manipulation, easy projection into 2D space, and to avoid estimating normals from SfM points. 3D Gaussians are defined by a position (mean) μ and a full 3D covariance matrix Σ. Projection to 2D for rendering is done via transformation using the Jacobian of the projective transformation. Instead of directly optimizing covariance matrices, they are presented as combining a scale and rotation matrix to ensure the covariance matrix stays positive and semi-definite.

Optimization with Adaptive Density Control of 3D Gaussian

3D Gaussian optimization tunes their parameters, positions, opacity (α), covariance, and spherical harmonics (SH) coefficients for color, and adaptively controls their density. The scene is rendered, and values are changed through Stochastic Gradient Descent (SGD). An initial sparse set of Gaussians is generated from SfM. Gaussians under a certain opacity are removed. Regions with high positional gradients are densified by cloning or splitting. To prevent unnecessary increases in Gaussian density, Gaussians are periodically culled.

Fast Differentiable Rasterizer for Gaussians

Use a tile-based rasterizer to pre-sort primitives for an entire image to avoid per-pixel sorting. The screen is split into 16x16 tiles where 3D Gaussians are culled against the view frustum and each tile. Each Gaussian is assigned a key that combines view space depth and tile ID. Gaussians are then sorted by their keys using GPU Radix sorting. During rasterization, one thread is launched per tile. Each pixel accumulates color and α values until a saturation (α) level is reached.

Implementation, Results, and Evaluation

Implementation was in Python using the PyTorch framework and wrote custom CUDA kernels for rasterization. NVIDIA CUB sorting routines were used for the fast Radix sort. An interactive viewer for measuring frame rates was built off of the open-source SIBR. A smaller image resolution is used initially, gradually increasing over several hundred iterations for optimization stability. SH coefficients optimization is sensitive to the lack of angular resolution (taking inside-out captures). To fix this, the zero-order SH component (diffuse color) is optimized first, and then after 1000 iterations, the next band of SH is introduced until all bands are represented.

Discussion and Conclusions

This paper establishes the first real-time rendering solution for radiance fields with comparable quality to high-cost methods while maintaining competitive training and rendering times.

Relightable 3D Gaussian Notes

Introduction

3D Gaussian Splatting (3DGS) surpasses all previous novel view synthesis solutions but cannot reconstruct a scene under different lighting conditions. Due to 3DGS rasterization, ray tracing's shadowing, and light reflectance require using ad hoc solutions (e.g., shadow maps, screen-space reflections, etc.). This paper proposes a novel 3D Gaussian point-based rendering that achieves physically based relighting. To make 3D Gaussians relightable, the following attributes are added: normal, BRDF properties, and incident light information. Incident light is split into a global environment map and an indirect incident light field. A novel ray tracing method based on bounding volume hierarchy (BVH). Regularizations, including constraints on depth distribution, smoothness priors, and a lighting regularization, are introduced to mitigate material-lighting ambiguity during optimization.

Relightable 3D Gaussians

3D Gaussian points are defined as:

$$G(x) = e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

$$\mu = \text{3D spatial mean}$$

$$\Sigma = \text{covariance matrix}$$

In the first step of the 3DGS rendering process, 3D Gaussians are projected to 2D Gaussians on the image plane. 3D spatial means are accurately projected to 2D means, but the 2D covariance matrices are approximated by:

$$\Sigma' = JW\Sigma W^T J^T$$

$$W = \text{View Transformation}$$

$$J = \text{Jacobian of the Affine Approximation of Perspective Transformation}$$

The pixel color is derived by alpha blending N ordered 2D Gaussians from front to back:

$$C = \Sigma_{i \in N} T_i \alpha_i c_i,\ T_i = \Pi_{j=1}^{i-1}(1-\alpha_j)$$

$$\alpha = o\Sigma'$$

In practice, Σ is parameterized as a unit quaternion (q) and a scaling vector (s). View-dependent color (c_i) is represented through a set of Spherical Harmonics (SH). Therefore, the i^{th} 3D Gaussian parameters is:

$$P_i = \{\mu_i, q_i, s_i, o_i, c_i\}$$

A normal attribute (n) is added to each 3D Gaussian. You cannot use the spatial mean of a 3D Gaussian as a conventional point and make normal estimations based on the local planar

assumption because the Gaussian point cloud is sparse and (more importantly) Gaussian points are not perfectly aligned with the object's surface. Therefore, n is initialized to a random vector via back-propagation and optimized. The pixel depth is estimated by the depth of all Gaussians along a ray. Additional densification on the gradient of normals is used to improve normal recovery in thin regions. A constraint minimizes the uncertainty of depth distribution to better align Gaussian points with the object's surface.

The PBR color is computed for each 3D Gaussian, and then alpha-blend them together to generate the PBR image. PBR properties are assigned to each Gaussian to make them relightable. To sample incident light at a Gaussian from direction w_i is represented as:

$$L_i(w_i) = V(w_i) \cdot L_{direct}(w_i) + L_{indirect}(w_i)$$
$$V(w_i) = \text{visibility term}$$

The indirect light term $L_{indirect}$ is parameterized by 3-level SH, and the direct light term L_{direct} is parameterized as a 16x32 environment map. Therefore, the i^{th} Gaussian is parameterized as:

$$P_i = \{\mu_i, q_i, s_i, o_i, c_i, n_i, b_i, r_i, l_i\}$$

Light Regularization is used to mitigate the materials-lighting ambiguity.

Point-based Ray Tracing

The novel point-based ray tracing approach is built upon the Bounding Volume Hierarchy (BVH). The binary radix tree is used as the algorithm for constructing a binary tree. Since Gaussians are semi-transparent, all Gaussian intersections must be considered. The ray travels from the camera center and accumulates transmittance as it passes through Gaussians until it reaches zero. To speed up ray tracing, there's an early return if transmittance drops below a certain threshold T_{min}.

The optimization process is divided into two stages. First, a 3D Gaussian point cloud with normal vectors is optimized. Then, the per-Gaussian visibility is pre-computed. Second, the geometry of 3D Gaussians is locked, and optimization is focused on the material and lighting parameters.

Experiments

The training is broken up into two stages. The first stage optimizes the proposed normal gradient-based densification, which consists of 30,000 iterations. The second stage optimizes the materials and lighting parameters, which consists of 10,000 iterations.

3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes Notes

Introduction

The critical contributions of the work are a GPU-accelerated ray-tracing algorithm for semi-transparent particles (Gaussians), an improved optimization pipeline for ray-traced, particle-based radiance fields, and generalized Gaussian particle formulations to reduce the number of intersections.

Related Work

To deal with secondary lighting effects (indirect lighting), the Relightable Gaussian paper uses spherical harmonics to encode occlusion information. Initial visibility rays determine occlusion, but ray tracing is restricted to the training phase. The Relightable Gaussian paper uses Axis-Aligned Bounding Boxes (AABBs), which are 3x slower than the stretched icosahedrons used in this paper. This paper also introduces a unique technique for tracing semitransparent particles (as most ray tracing is highly optimized for rendering opaque surfaces).

Background

Kernel function of a 3D Gaussian particle:

$$p(x) = e^{-(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

μ = particle's position

$$\Sigma = RSS^T R^T$$

R = rotation matrix

S = scaling matrix

Use a spherical harmonics function to determine the opacity coefficient (which depends on the view direction).

Method

Simply intersecting a ray with the AABB around each particle is fast, but a diagonally stretched Gaussian particle will cause the traversal to evaluate many false-positive intersections. Stretched Polyhedron Proxy Geometry (icosahedron) bounding volumings were the most performant.

Volumetric rendering requires accumulating the contribution of particles along the ray in a *sorted* order. Their renderer gathers the next k (where k = 16 gives good results) particles, maintains a sorted buffer of their indices, then iterates through the sorted array of primitive hits, retrieves the corresponding particle for each, and renders them according to the following equation:

$$\boldsymbol{L}(\boldsymbol{o},\boldsymbol{d}) = \sum_{i=1}^{N} \boldsymbol{c}_i(\boldsymbol{d})\alpha_i \prod_{j=1}^{i-1} 1 - \alpha_j,$$

The process repeats, tracing a new ray from the last rendered particle to gather the next *k* particles. The process terminates once all particles intersecting the ray are processed or when enough particle density is intersected.

Rather than using standard 3D Gaussian kernels, described by the following equation:

$$p(x) = e^{-(x-\mu)^T\Sigma^{-1}(x-\mu)},$$

degree-2 generalized Gaussian particles (GG2), described by the following equation:

$$p(x) = e^{-((x-\mu)^T\Sigma^{-1}(x-\mu))^n}$$

$$n = 2,$$

defines denser particles which reduces the number of intersections and increases the performance by a factor of 2.

Experiments and Ablations

Use degree-2 generalized Gaussian particles, a density learning rate of 0.09 during optimization, and optimizing with incoherent rays in a batch size 2^{19} starting after 15,000 training iterations. Evaluation scenes used: four indoor (room, counter, kitchen, bonsai), three outdoor (bicycle, garden, stump), and two large outdoor scenes (truck and train).

The best bounding primitive was icosahedron: a twenty-faced regular polyhedron mesh.

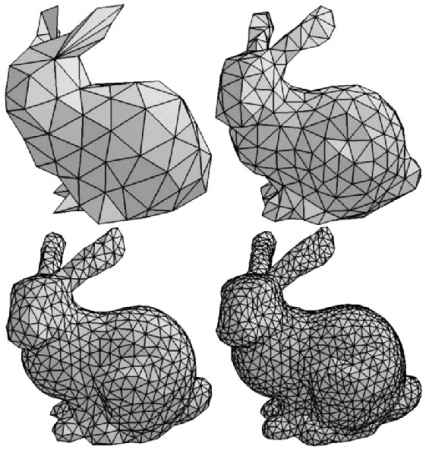
The best tracing algorithm was their proposed method. Their method + 2x2 tiled tracing, where tracing one ray per 2x2 tile but still evaluating appearance per pixel, akin to tile-based rasterization, produced slightly worse visual results with significantly better performance.

Applications

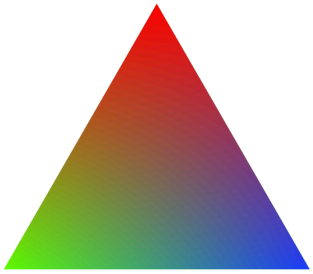
They maintain an extra acceleration structure consisting only of mesh faces for additional inserted geometry. When casting each ray, they first cast rays against inserted meshes. If a mesh is hit, render all particles only up to the hit and compute a response based on the material. For refractions and reflections, this means continuing tracing along a new redirected ray according to the laws of optics. For non-transparent diffuse meshes, we compute the color, blend it with the current radiance, and then terminate the ray.

Traditional 3D Polygon Models

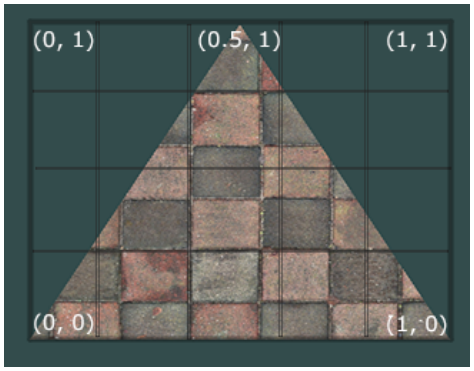
A series of connected triangles approximate the shape of real-life 3D objects. Triangles are used because it is the minimum number of points to describe a plane, a.k.a. face, in three dimensional space.



Each vertex, i.e., points of a triangle, holds its (x,y,z) positional data. Each vertex can have an arbitrary number of attributes. These vertex attributes get linearly interpolated across the face of the triangle. For example, you can assign the colors red, green, and blue to the corners of a triangle.



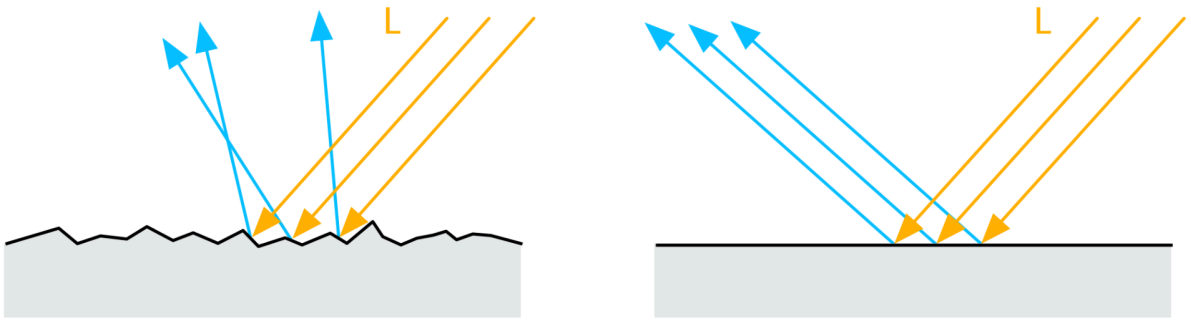
If we wanted to overlay an image across the triangle, we would give each vertex a UV coordinate that indexes the image.



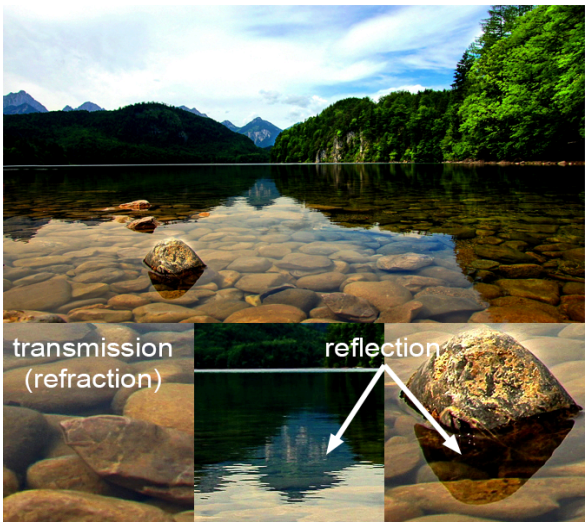
Physically Based Rendering (PBR)

Physically-based rendering follows three rules: energy conservation, the microfacet model, and the Fresnel Effect.

A microfacet model is where object surfaces at a microscopic level are composed of jagged faces that perfectly reflect light; surface misalignment results in the surface appearing rough at a macroscopic level. The less rough a surface is, the more it perfectly reflects light.

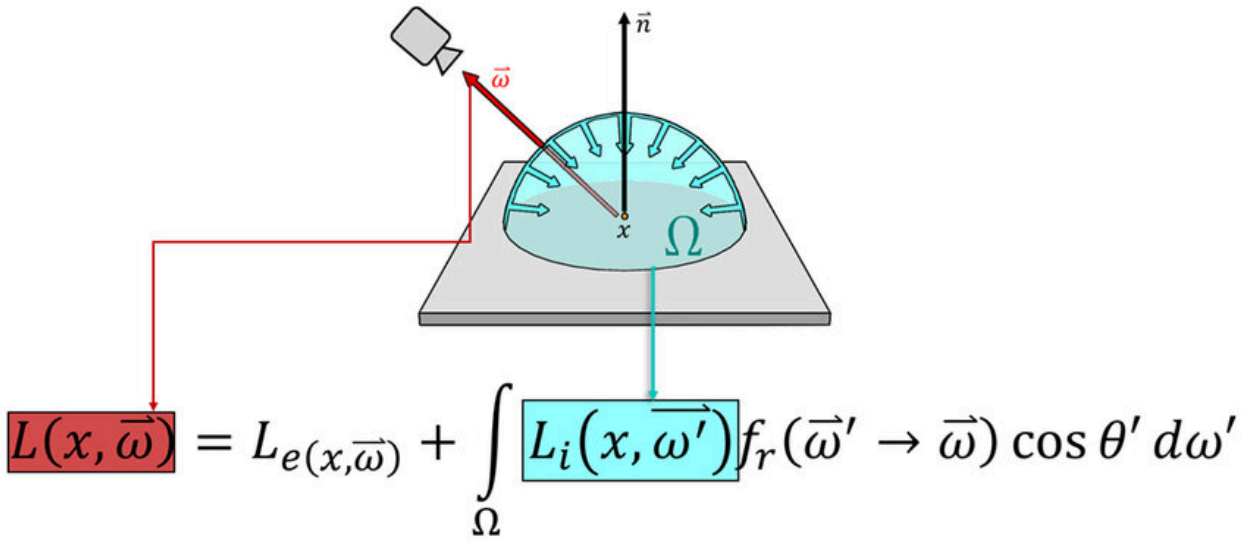


The Fresnel Effect is the phenomenon where the lower the viewing angle on a surface is, the better the reflection is.



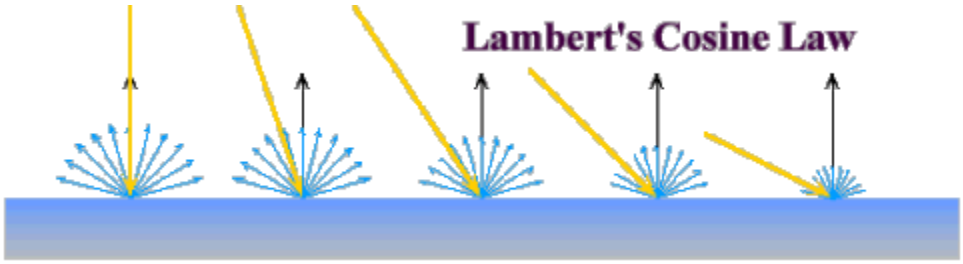
The Rendering Equation

The rendering equation calculates the light (radiation waves in the visible spectrum) we see at a particular point. This equation states that the radiance at a point equals the sum of the emitted radiance at the point and the sum of all the incoming radiance reflected at a single point.



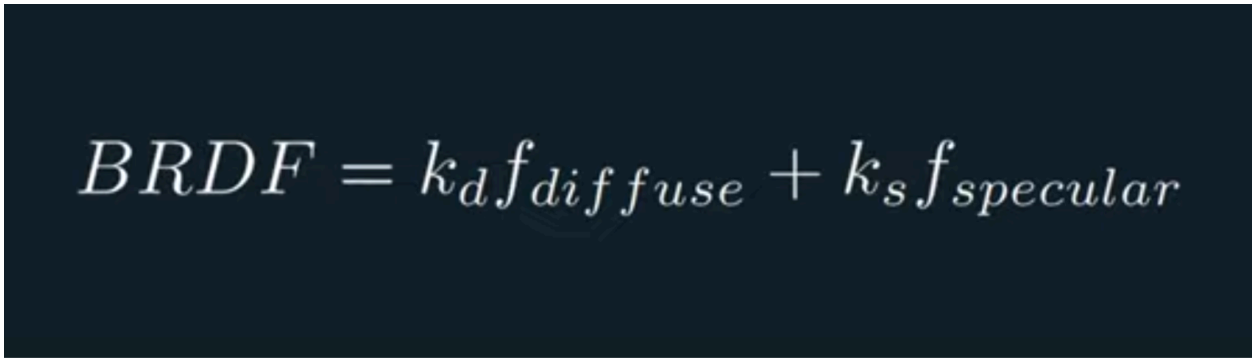
L(x,w) is your outgoing light at a single point, x, given your viewing direction w. L_e (x,w) is the emitted light. L_i(x,w') is your incoming light value. This would be the color for directional lights, but you must account for light falloff for point and spotlights.

The 'cosθ' portion of the integral is Lambert's Law, which describes the relationship between the light direction, denoted by L, and a surface's normal, denoted by N. Light gets stretched across a larger area as it approaches more grazing angles. The surface gets darker as L and N become perpendicular, and vice versa. This behavior is encoded by the cosine of the angle between L and N, calculated by dot(L, N).



Bidirectional Reflectance Distribution Function (BRDF)

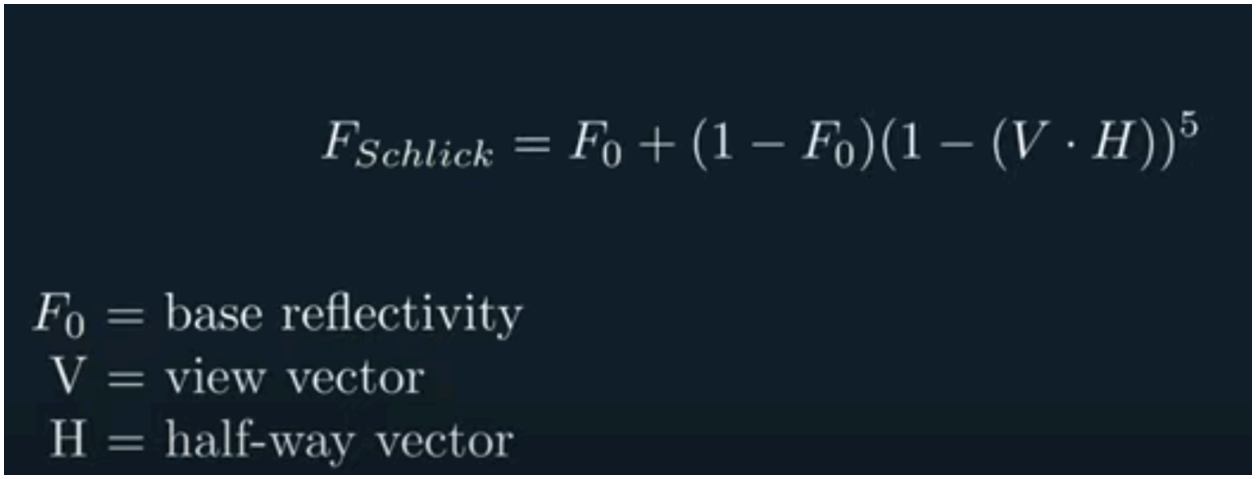
The fr(w'→w) portion of the integral is called the Bidirectional Reflectance Distribution Function (BRDF), which describes how much light gets reflected off a surface.



The BRDF equation is the sum of the diffuse and specular lighting, while kd and ks represent what fraction of the lighting types contribute. Ks = Fresnel Function, and, to conserve energy, kd = 1 - ks.

Fresnel Function

The following is Schlick's Fresnel approximation, often used for the Fresnel function. The half-way vector is the halfway point between the view and the light vector. The base reflectivity of a dielectric material is a constant value of 0.04 (most dielectric materials have a based reflectivity between 2%-5%), while the base reflectivity of a metal is the albedo.



Diffuse Function

The Lambertian Diffuse BRDF is used for the diffuse function, simply the albedo of your object divided by π.

$$f_{Lambert} = \frac{color}{\pi}$$

Specular Function

The Cook-Torrance (a.k.a. Torrance-Sparrow) equation is commonly used as the specular function. The normal distribution function is represented by D, G represents the geometry shadowing function, and the fresnel function is represented by F.

$$f_{Cook-Torrance} = \frac{DGF}{4(V \cdot N)(L \cdot N)}$$

Normal Distribution Function

The industry's most common normal distribution function is the GGX/Trowbridge-Reitz model.

$$\alpha = roughness^2$$

$$D_{GGX/Trowbridge-Reitz} = \frac{\alpha^2}{\pi((N \cdot H)^2(\alpha^2 - 1) + 1)^2}$$

Geometry Shadowing Function

Again, the Schlick-Beckman model is the most common function used for geometry shadowing. You must multiply the G(L) by G(V) to get your G value.

$$G_{Schlick-Beckmann} = \frac{N \cdot X}{(N \cdot X)(1-k)+k}$$

$$k = \frac{\alpha}{2}$$

Metallic Rendering Special Case

Metals only reflect their specular highlight. Therefore, simply multiply the kd variable by one minus the metallic value to render metallic materials.

Expanded PBR Rendering Equation

This puts the rendering equation in terms of color, roughness, metalness, and normal direction.

$$L_o(x,V) = L_e(x,V) + \int_\Omega \left(k_d \frac{color}{\pi} + k_s \frac{DG}{4(V \cdot N)(L \cdot N)} \right) L_i(x,L)(L \cdot N) dL$$

$$k_s = reflectivity + (1 - reflectivity)(1 - (V \cdot H))^5$$

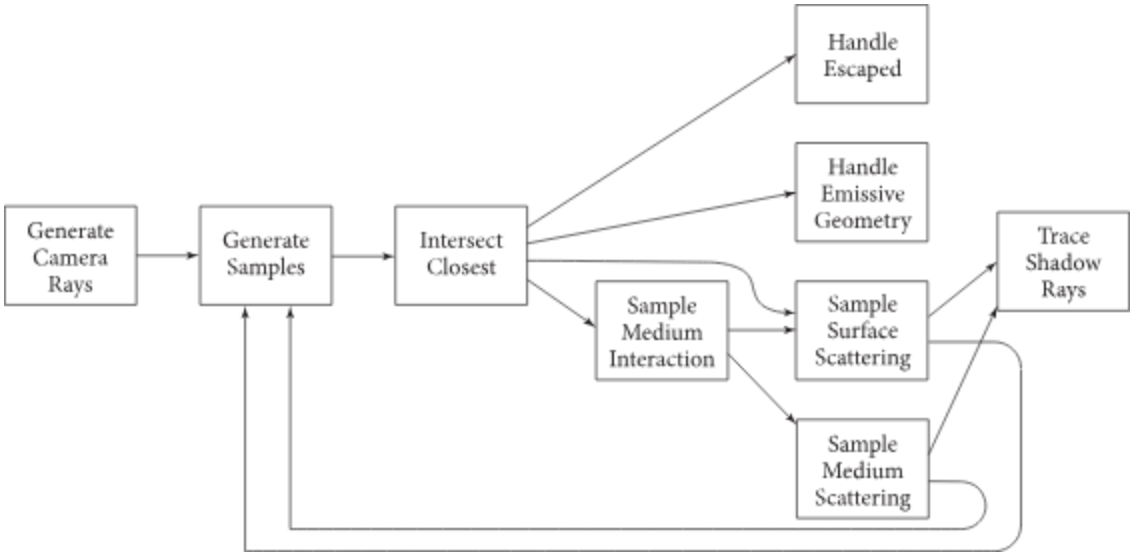
$$k_d = 1 - k_s$$

$$D = \frac{roughness^2}{\pi((N \cdot H)^2(roughness^2 - 1) + 1)^2}$$

$$G = \left(\frac{N \cdot L}{(N \cdot L)(1 - \frac{roughness}{2}) + \frac{roughness}{2}} \right) \left(\frac{N \cdot V}{(N \cdot V)(1 - \frac{roughness}{2}) + \frac{roughness}{2}} \right)$$

Raytracing

Path Buffers:
path paths[viewport.width * viewport.height * pathsPerPixel]
pathId pathsContinue[paths.size() + 1], where the last index contains the count
pathId pathsEnd[paths.size() + 1], where the last index contains the count



Bounding Volume Hierarchy (BVH) Construction

Construct a binary radix tree for the 3D Gaussian models (4.1 Relightable Gaussian paper).

Axis-Aligned Bounding Boxes (AABBs) are the industry standard choice for Bounding Volume Hierarchies (BVHs) due to their ease of use and performance. Surface Area Heuristic (SAH) is the most popular method for constructing highly performant BVHs. It uses a cost model based on the surface areas of the bounding boxes and the number of primitives within them to determine where to split using the following equation:

$$C(A,B) = t_{traversal} + p_A \sum_{i=1}^{N_A} t_{intersect}(a_i) + p_B \sum_{i=1}^{N_B} t_{intersect}(b_i)$$

Primary Ray Generation

For a simple pinhole camera, convert pixel index coordinates (x, y), where $x \in \{0, 1, 2, \dots, ImageWidth - 1\}$, $y \in \{0, 1, 2, \dots, ImageHeight - 1\}$, into a point on the image plane in camera coordinates, where $x \in [-1, 1]$, $y \in [-1, 1]$, using the following equations:

$$ImagePlane_x = \frac{2x+1}{ImageWidth} - 1,$$
$$ImagePlane_y = \frac{-2y-1}{ImageHeight} + 1$$

We must linearly interpolate x to account for the image's aspect ratio such that:

$$x \in [-AspectRatio, AspectRatio],$$

$$\text{where } AspectRatio = \frac{ImageWidth}{ImageHeight}$$

Using the following equation:

$$ImagePlane_x = \left(\frac{2x+1}{ImageWidth} - 1\right) * \frac{ImageWidth}{ImageHeight} = \frac{2x+1}{ImageHeight} - AspectRatio,$$

Let d represent the distance between the camera and the image plane (along the z-axis). It is expected to keep the focal length equal to 1 for simplicity. *NOTE: Unity uses a left-handed, y-up coordinate frame.* To account for the Field Of View (FOV), we need to scale $ImagePlane_x$, $ImagePlane_y$ appropriately with the following:

$$ImagePlane_x = \left(\frac{2x+1}{ImageHeight} - AspectRatio\right) * d * tan(\frac{FOV}{2}),$$
$$ImagePlane_y = \left(\frac{-2y-1}{ImageHeight} + 1\right) * d * tan(\frac{FOV}{2}),$$
$$ImagePlane_z = d$$

We can then transform this point into world coordinates using a camera-to-world matrix multiplication. The camera-to-world matrix is defined by a look-at rotational matrix, denoted by R, then a translation, denoted by T, $(T \times R)$.

Ray-Box Intersection

We can define an AABB by two points of its corners:

$$p_{min} = \begin{bmatrix} x_{min} \\ y_{min} \\ z_{min} \end{bmatrix}, p_{max} = \begin{bmatrix} x_{max} \\ y_{max} \\ z_{max} \end{bmatrix}$$

A point described as:

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

, lies inside the AABB if and only if $p_{min} < p < p_{max}$, i.e.:

$$x_{min} < x < x_{max}$$
$$y_{min} < y < y_{max}$$
$$z_{min} < z < z_{max}$$

p can be described using the ray's parametric equation:

$$p = o + td \Rightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} + t \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$$

where o is the ray's origin and d is the ray's direction. Therefore, we can determine where the ray intersects an axis-aligned plane with the following:

$$f(x) = t = \frac{x - x_o}{x_d}$$

The intersection of three segments is the following:

$$t_{min} = max(f(x_{min}), f(y_{min}), f(z_{min}))$$
$$t_{max} = min(f(x_{max}), f(y_{max}), f(z_{max}))$$

Where if $t_{min} > t_{max}$, then there is no intersection.

Ray-Triangle Intersection

Möller-–Trumbore is the industry standard ray-triangle intersection algorithm. The intersection point on a triangle (P) can be described with the following equation:

$$P = wA + uB + vC$$

Where $w + u + v = 1, w, u, v \in [0, 1]$, and A, B, C are the vertices of the triangle. We can rewrite this equation using only two two coefficient terms, u, v.

$$P = (1 - u - v)A + uB + vC$$
$$\Rightarrow P = A + uB - uA + vC - vA$$
$$\Rightarrow P = A + u(B - A) + v(C - A)$$

P can equivalently be described using the ray's parametric equation:

$$P = O + tD$$

where O is the ray's origin and D is the ray's direction. Using substitution, we can rewrite our original equation to the following:

$$O + tD = A + u(B - A) + v(C - A)$$
$$\Rightarrow -tD + u(B - A) + v(C - A) = O - A$$

$$\Rightarrow [-\ D\ \ (B\ -\ A)\ \ (C\ -\ A)][t\ \ u\ \ v]^T = O\ -\ A$$

We can use Cramer's rule, which states:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, A_x = \begin{bmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{bmatrix}, A_y = \begin{bmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{bmatrix}, A_z = \begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{1}{det(A)} \begin{bmatrix} det(A_x) \\ det(A_y) \\ det(A_z) \end{bmatrix}$$

, and scalar triple product rule, which states:

$$det(A) = det\left(\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}\right) = det\left(\begin{bmatrix} x & y & z \end{bmatrix}\right) = (x \times y) \cdot z$$

, to solve for $[t\ \ u\ \ v]^T$, giving us:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times (C - A)) \cdot (B - A)} \begin{bmatrix} ((O - A) \times (B - A)) \cdot (C - A) \\ (D \times (C - A)) \cdot (O - A) \\ ((O - A) \times (B - A)) \cdot D \end{bmatrix}$$

Ray-Gaussian Intersection

3D Gaussians are essential fuzzy, blob-like points in space (think of a blurry ellipsoid or cloud). Since a Gaussian is semi-transparent, you cannot calculate the exact point of intersection. Instead, you find a point along the ray where the Gaussian's influence peaks by calculating where the ray comes closest to the center of the Gaussian, given its spread and orientation. For each Gaussian a ray passes through, it reduces the ray's transmittance. The ray stops once its transmittance has dropped below a certain threshold (4.1 Relightable Gaussian paper).

Glossary

Term	Definition
Face	The surface of a triangle.
Fresnel Effect	The change in how much light reflects off a surface is based on the viewing angle.
Microfacet Model	A way to approximate a surface as a collection of small, individual faces called microfacets.
Normal	A directional vector that is perpendicular to the face of a triangle.
Novel View Synthesis	Generate new images of a scene or object from a specific viewpoint when the only available information is pictures taken from different perspectives.
UV Coordinates	2D coordinates map a texture onto a 3D model's surface.
Vertex	The point where two edges of the triangle meet.

Resources

- 3D Gaussian Splatting for Real-Time Radiance Field Rendering
- 3D Gaussian Splatting Resources (GitHub)
- Adobe The PBR Guide
- Computer Graphics Tutorial - PBR (Physically Based Rendering)
- Demystifying Floating Point Precision
- Optimizing a Ray Tracer (by building a BVH)
- Physically Based Rendering (book)
- Relightable 3D Gaussian: Real-time Point Cloud Relighting with BRDF Decomposition and Ray Tracing
- Understanding Radiance (Brightness), Irradiance, and Radiant Flux
- Möller-Trumbore Algorithm
- Fast, Branchless Ray/Bounding Box Intersections, Part3: Boundaries
- Generating Camera Rays with Ray-Tracing
- How to Create Awesome Accelerators: The Surface Area Heuristic