Hybrid Relightable 3D Gaussian Rendering: A Real-Time Novel View Synthesis System with Triangle Mesh Integration

By Jackson Vanderheyden, Brian Xicon, Ethan Gasner, Luke Broglio, Kyle Kohl

Table of Contents

1. Introduction

1.1 Problem Statement

The technique of generating images of a scene from any perspective using only a video or a series of photographs is known as novel view synthesis. Previously, this was done using Neural Radiance Fields (NeRFs) or photogrammetry. Still, these methods have costly training and rendering times, can produce subpar results, are unable to generate images under different lighting conditions, and cannot be combined with traditional polygon meshes.

Our solution is to create a novel view synthesis system using a new technique called 3D Gaussian Splatting (3DGS). This solution is significantly faster and creates higher-quality renders than NeRFs, but two lingering issues remain. Standard 3DGS models are not relightable, meaning that the renders cannot have different lighting conditions than those in the input video or images, and the 3D Gaussian renderer is incompatible with industry-standard polygon meshes.

To solve these issues, we plan to build off the work of relightable Gaussians, which uses a machine learning model to extract the model's lighting and material properties and create a new ray-traced rendering system that seamlessly integrates triangle and Gaussian models. This system will allow users to change the scene's lighting in real time, take advantage of global illumination features such as reflections and shadows, and add traditional polygon meshes into the scene.

This system will be beneficial in various circumstances, including game development and 3D art, as part of sales to allow customers to see an accurate and high-quality depiction of a product and as an improvement for current NeRFs or photogrammetry systems.

1.2 Intended Users and Uses

We believe three primary personas accurately represent the individuals who will use our product: "NeRF User," "Non-Technical User," and "3D Artist."

The "NeRF User" persona actively performs novel view synthesis using existing technology such as NeRFs. This user group comprises people familiar with 3D rendering technology and novel view synthesis. "NeRF Users" seek better-performing view synthesis techniques to train and efficiently generate high-quality renders to improve their workflow. Our solution appeals to this group because of our system's performance improvements over their current tools. It also offers new functionality, allowing them to do what they already do faster and with a higher-quality end product.

The second persona, the "Non-Technical User," wants to display a scene or an object but cannot due to the technical knowledge barrier for current view synthesis models. For example, a "Non-Technical User" may be a realtor who wants to allow potential home buyers to view a property they have a video of as a 3D render, or they could be an owner of a furniture store wanting to visualize their products for their customers better. This user group must easily create and display an object or scene in three-dimensional space. Our product

aims to be a user-friendly solution to novel view synthesis to ensure an enjoyable user experience for technical and non-technical users, making it incredibly appealing to "Non-Technical Users."

The last user persona is the "3D Artist". This person is the user who already has some knowledge of 3D rendering and is creating traditional triangle meshes using 3D modeling software such as Blender, Cinema4d, or Maya. While this user will already know about 3D rendering, they may not know of novel view synthesis. Currently, "3D artists" must recreate objects or scenes by hand, which is incredibly time-consuming. These users seek new technologies to improve their workflow that won't inhibit current industry norms. To best accommodate this user group, we want our solution to be easily integrated with existing 3D rendering technologies and tools and usable by individuals without preexisting novel view synthesis knowledge. Our solution will directly appeal to this type of user by allowing them to efficiently create models and scenes from videos rather than making them by hand.

2. Requirements, Constraints, And Standards

2.1 Requirements & Constraints

2.1.1 Ubiquitous Functional Requirements

- The system shall input a Structure from Motion (SfM) point cloud for the relightable Gaussian model generator. (constraint)
- The system shall take a relightable Gaussian model, which is composed of incident light, normal direction, Bidirectional Reflectance Distribution Function (BRDF) properties, position, opacity, anisotropic covariance, and spherical harmonics, as input for the 3D Gaussian parser. (constraint)
- The system shall input a parsed 3D Gaussian model for the Axis-Aligned Bounding Box (AABB) Bounding Volume Hierarchy (BVH) generator. (constraint)
- The system shall take Unity mesh objects as input for the AABB BVH generator. (constraint)
- The system shall take relightable 3D Gaussian models with constructed AABBs, triangle meshes with constructed AABBs, scene point, spot, and directional lighting data, and camera parameters as input for the hybrid Gaussian and triangle mesh ray tracer. (constraint)
- The system shall take the final rendered texture from the hybrid Gaussian triangle mesh ray tracer as input for the Unity Camera's rendered texture. (constraint)

2.1.2 Event-Driven Functional Requirements

- When rendering the next frame is initiated, the system shall place all paths into the ended paths buffer.
- When the primary ray generation compute shader is called, and the ended paths buffer contains paths, the system shall generate a new origin and ray direction to align the path with its associated pixel for each path in the ended paths buffer.
- When the determined path intersection is computed shader is called, and the ended paths buffer contains paths. The system shall calculate ray intersections given the path direction. If the path hits a relightable Gaussian model or a triangle mesh, add the path to the continuing paths buffer; otherwise, add the path to the ended paths buffer.

- When the lighting calculation computes shader is called, and the continuing paths buffer contains paths, the system shall determine how much light the surface absorbs at the intersection point and calculate the reflection direction. If the path has reached its max bound limit, don't let it continue.

2.1.3 State-Driven Functional Requirements

- Initial State: Gaussians are initialized with positions and default normals; however, no optimization is performed on the normals or the material's properties.

- Intermediate State: The PyTorch Gaussian optimizer shall adjust the normals to align with the lighting environment.

- Final State: After optimizing the normal parameters, the system shall enable BRDF property optimization. This includes properties like albedo and roughness. This will allow for realistic material behavior with the relighting.

2.1.4 Look & Feel Nonfunctional Requirements

- The system should be available as a package for the Unity Game engine. It should be able to be added to existing Unity projects and work with the existing tools provided by Unity. (constraint)

2.1.5 Usability Nonfunctional Requirements

- The system should be easy to use with the tools and systems provided by the Unity engine and integrate naturally with the rest of the system.

- The system should be intuitive and self explanatory so it's usable by someone without a technical or graphics programming background.

2.1.6 Performance Nonfunctional Requirements

- The system should run in real-time. This means that the system should allow the user to change the scene's lighting or the location of the camera's angle, and the system should update the scene dynamically at 30 FPS. (constraint)

- The system should produce high-quality renders that don't appear fake to the user, i.e., renders don't contain glaring visual artifacts like noise, shadow ripples, or erroneous sampling.

2.1.7 Operational Nonfunctional Requirements

- The system should be available for download as a Unity package using Unity's package manager. (constraint)

2.2 Engineering Standards

Several key frameworks may apply when considering engineering standards for our project, including IEEE/ISO/IEC 12207-2017, ISO/IEC 23053:2022, and ISO/IEC 23488:2022. First, standard IEEE/ISO/IEC 12207-2017 outlines a comprehensive framework for software lifecycle processes, detailing best practices for software development, maintenance, and management. This standard ensures well-defined software processes, promoting consistency and quality throughout the development lifecycle. Following this standard will help our team ensure that our project constantly evolves and improves weekly.

We will integrate the IEEE/ISO/IEC 12207-2017 framework into our development process. This will include establishing precise requirements, design, implementation, testing, and maintenance throughout development. We will also conduct regular reviews and team updates via the agile development process. This will ensure that we adhere to the standard through all stages of development.

ISO/IEC 23053:2022 provides guidelines for assessing AI systems that use machine learning tactics, focusing on their reliability and performance. As AI plays a vital role in many projects, following this standard is crucial to ensure that the systems developed are effective and trustworthy. By implementing the principles in this standard, our team can better evaluate the robustness of our Gaussian point optimizer and our material prediction machine learning models; this will enhance user confidence in our final deliverable.

To comply with standard ISO/IEC 23053:2022, we plan to develop a comprehensive evaluation for our generated machine learning models, specifically the Gaussian point optimizer and the material prediction models. When creating our evaluation framework we will perform performance metrics to easily allow us to check the reliability and effectiveness of our models. We will perform these metrics throughout the model's development cycle.

Finally, ISO/IEC 23488:2022 focuses on representing objects and environments for image-based rendering in virtual, mixed, and augmented reality (VR/MAR). This standard provides clear rules for capturing and showing the visual details of real and virtual objects, which helps create high-quality, immersive experiences. By outlining a way to represent objects and environments, the standard will help our team make the dynamic raytracer that acts as one of the center points of our project. Following this standard will ensure our applications work well with current and upcoming software. Using ISO/IEC 23488:2022 will help improve the visual quality of our virtual environments and also help create more exciting and realistic interactions within our virtual environments.

We will remain consistent with the ISO/IEC 23488:2022 standard by incorporating the best practices for rendering using our dynamical raytracer. This will include clearly defining our techniques for object representation and ensuring that all objects are rendered as faithfully to their original forms as possible. We will use the guidelines specified within the standard to help enhance the visual quality of our environments and create realistic interaction between our light source and rendered objects.

3 Project Plan

3.1 Project Management/Tracking Procedures

Our team is using an agile workflow with 2-week sprints. We chose agile because we will use progressive elaboration to develop our project plan further as we learn more about the technologies and techniques used in Gaussian Splatting. Working in 2-week sprints will allow us to make the most out of this process.

For communication and scheduling, we are using Discord. We are tracking tasks using the Gitlab issue board and using Git and branches to allow us to work collaboratively on the project. Finally we are using our weekly reports to document our progress so our advisor (or anyone else) can follow our work from the outside.

3.2 Task Decomposition

Hybrid Gaussian Raytracer:

- Using PBR techniques, generate a render of composite scenes with triangles and Gaussian models.

- Get Scene Data
    - Get Unity camera parameters
    - Get scene lighting information
        - Directional light support
        - Spotlight support
        - Point light support
    - Get triangle mesh objects in the scene
        - Generate Axis-Aligned Bounding Boxes using surface area heuristic.
    - Get Gaussian objects in the scene
        - Parse Gaussian splat .ply files
        - Generate Axis-Aligned Bounding Boxes using surface area heuristic.
- Create a rendering pipeline.
    - Create a command buffer containing all necessary rendering commands and insert it in the camera's rendering pipeline stage.
        - Preprocess all path buffer information.
            - Fill a path ends buffer with all of the paths
            - Clear the paths continue buffer
            - Clear the temporary paths continue buffer
            - Fill the camera information buffer with current camera parameters
        - Generate all primary rays for paths in the path ends buffer and place them into the paths continue buffer
        - Loop through all paths in the paths continue buffer, and determine if there is an intersection. If there is, add the path to the temporary paths continue buffer; otherwise, add the path to the paths end buffer.
        - Loop through the temporary paths continue buffer, and do shading calculations. Place these paths back into the (non-temporary) paths continue buffer.
        - Repeat the intersection and shading code until paths have reached their bounce limit.

Machine Learning Models:

- Data Preparation
    - Gather a dataset of point clouds with known PBR properties (e.g., albedo, roughness, metallic).
    - Preprocess point clouds (e.g., scaling, outlier removal).
    - Divide the dataset into training, validation, and testing sets.
- Feature Extraction
    - Identify relevant features for PBR properties (e.g., surface geometry, texture gradients).
    - Develop a method to extract these features from the point set.
- Model Selection
    - Select a suitable ML architecture (regression models using SGD).
    - Define input-output mapping (input: point cloud; output: PBR properties).
- Training the Model
    - Choose an appropriate loss function (e.g., Mean Absolute Error for property estimation).
    - Set up the training configuration (optimizer, epochs).
    - Train the model using the training dataset and validate using the validation set.

- Model Evaluation
    - Evaluate the trained model on the testing dataset.
    - Calculate performance metrics
    - Review results and refine the model or input features if needed.
- Integration and Deployment
    - Integrate the PBR property extraction model into the rendering pipeline.
    - Develop an interface for inputting point clouds and outputting PBR properties.
    - Optimize the model for speed and efficiency in real-time applications.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Graphics / Raytracer Milestones

- Path tracer with triangle support
    - The program should render a scene or object defined using traditional triangle meshes using a path tracer.
    - Performance should be at least 30 FPS
    - Takes in Unity scene data
- Path tracer with material support
    - The path tracer correctly renders triangles with materials attached instead of simple colors.
    - Performance should be at least 30 FPS
- Path tracer with basic Gaussian support
    - The path tracer should now allow rays to intersect with one or multiple 3d Gaussians to determine the color of a pixel.
    - Performance should be at least 30 FPS
    - The program should be able to read Gaussian data from a file.
- Path tracer with normal Gaussians (Gaussian with normal directional vectors) support
    - Using their normal vector, the path tracer should now utilize reflection and scattering rays to determine color when intersecting with Gaussians.
    - Performance should be at least 30 FPS
    - The files containing Gaussians should contain the normal vectors
- Path tracer with relightable Gaussian support
    - The path tracer should now render Gaussians using their Physically based rendering properties.
    - The color of the pixels when intersecting Gaussians should now be responsive to changes in lighting.
    - Performance should be at least 30 FPS.

Machine Learning Milestones

- Data Preparation and Feature Extraction
    - Gather and preprocess the dataset of point clouds with known PBR properties.-
    - Identify and extract relevant features for PBR property estimation.
    - Deliverable: A cleaned dataset and a documented feature extraction method.
- Model Development and Training
    - Select the ML architecture and define input-output mappings.
    - Train the model using the training dataset and validate it with the validation set.
    - Deliverable: A trained model(s) with validation results.
- Evaluation and Integration

- Evaluate the model on the testing dataset, refine it as needed, and integrate it into the rendering pipeline.
- Develop an interface for inputting point clouds and outputting PBR properties.
- Deliverable: A functioning model integrated into the pipeline, performance metrics, and user interface.

3.4 Project Timeline/Schedule

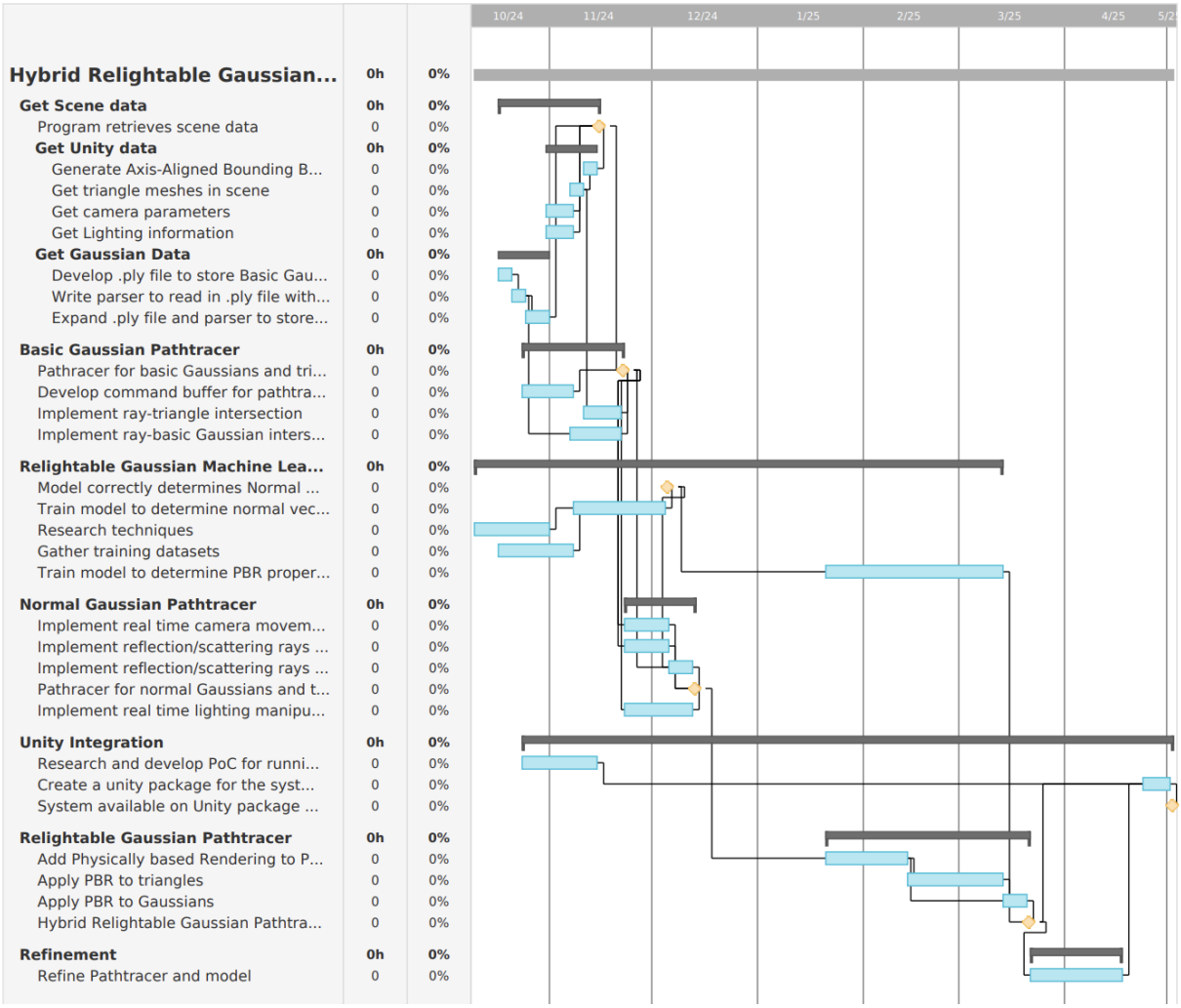

Figure 1: A projected Gantt chart for the extent of the project.

Note: This chart will change as the semester progresses because we are incorporating progressive elaboration into our development process. Tasks later in the process will be broken down into more detail.

In summary, our project is broken into four different deliverables.

First, we will create a basic path tracer supporting triangles and basic Gaussians. This will not require anything from the machine learning component but does require the tasks under **Get Scene Data** to be completed.

The following deliverable is the normal Gaussian path tracer. This will improve the path tracer to render scenes using reflection and scattering rays. To do this, we must modify the pathtracer by completing the tasks under the **Normal Gaussian Pathtracer** group. In addition, the path tracer will need the Gaussian files it reads in to contain normal vectors for all the Gaussians. The machine learning model will determine these vectors, so the **Train model task will determine the normal vectors needed in the normal vectors task**.

The third deliverable is the full Hybrid Relightable Gaussian Pathtracer. In this deliverable, the Pathtracer should render scenes using Physically Based Rendering. The process to implement this will be similar to the process of moving from the basic to the normal Gaussian path tracer. The pathtracer must be expanded to support physically based rendering for triangles and Gaussians by completing the tasks within the **Relightable Gaussian Pathtracer** group. The machine learning model will then need to be trained to determine the Gaussian PBR properties by completing the **Train model to determine the PBR properties** task.

For the final deliverable, we will make any necessary adjustments and refinements to the path tracer and the model to finalize them. Then, we will package them together in a Unity package so the system can be distributed and reused.

Milestone	**Expected Completion Date**
The program retrieves scene data from Unity scene and Gaussian files	11/15/24
Pathtracer for triangles and Basic Gaussians	11/22/24
The model correctly determines normal vectors for Gaussians	12/5/24
Pathtracer for triangle and Gaussians with normal vectors	12/19/24
Pathtracer for triangles and Relightable Gaussians with Physically-based Rendering	3/21/25
System available on Unity package manager	4/26/25

Figure 2: Milestone goals for the extent of our senior design project.

3.5 Risks And Risk Management/Mitigation

When developing a complex system that combines a Hybrid Gaussian Raytracer with machine learning models for PBR property extraction, several risks can arise throughout the project lifecycle. Identifying and managing these risks is crucial for ensuring successful outcomes and minimizing disruptions.

One of the primary risks is the technical complexity of integrating the raytracing system with the machine learning model. Differences between the data formats used in the rendering pipeline and those required by the ML model could lead to compatibility issues. Early prototypes should be developed to test the data interchange processes to mitigate this risk. Establishing clear interfaces and using standardized formats can advocate for smoother integration. Continuous communication between the teams working on the rendering engine and the machine learning components will also help identify potential issues early.

The success of the machine learning model heavily relies on the quality and representativeness of the training dataset. Risks include insufficient diversity in the point clouds, noise, and outliers that could skew the model's performance. A rigorous data collection and preprocessing pipeline should be implemented to manage this risk, including outlier detection and handling techniques. Regular dataset validation against real-world scenarios will ensure it remains relevant and robust.

Finally, once integrated, the system must be optimized for performance in real-time applications. Risks in this phase include performance degradation due to inefficient algorithms or inadequate hardware resources. Regular profiling of the rendering pipeline and machine learning components can help identify performance bottlenecks early. Additionally, incorporating optimization techniques, such as reducing the complexity of algorithms and leveraging parallel processing, will enhance the system's efficiency.

3.6 Personnel Effort Requirements

Task	**Person-Hours**
Generate Axis-Aligned Bounding Boxes using surface area heuristic.	6
Get triangle meshes in the scene	4
Get camera parameters	2
Get Lighting information	2
Develop a .ply file to store Basic Gaussians	6
Write parser to read in .ply file with Gaussians	3
Expand .ply file and parser to store Relightable Gaussians	4
Develop command buffer for path tracer	10
Implement ray-triangle intersection	8

Implement ray-basic Gaussian intersection	10
Train model to determine normal vectors from Basic Gaussians	24
Research machine learning techniques	20
Gather training datasets	10
Train model to determine PBR properties for Gaussians	36
Implement real-time camera movement	10
Implement reflection/scattering rays for triangles	16
Implement reflection/scattering rays for Gaussians	16
Implement real-time lighting manipulation	10
Research and develop PoC for running Pytorch inside Unity	20
Create a unity package for the system	10
Add Physically-based Rendering to Pathtracer	24
Apply PBR to triangles	12
Apply PBR to Gaussians	12
Refine Pathtracer and model	24

Figure 3: Projected task decomposition and their associated developmental hours estimate.

Note: These estimates will change as the semester progresses because we are incorporating progressive elaboration into our development process. Tasks later in the process will be broken down into more detail.

3.7 Other Resource Requirements

It is crucial to secure adequate computational power to successfully develop and deploy our hybrid Gaussian raytracer and machine learning models. The rendering engine and machine learning components are resource-intensive, requiring significant processing capabilities to efficiently handle complex calculations and large datasets. The rendering engine must support real-time performance while processing high-quality graphics, which demands powerful GPUs and ample memory.

Simultaneously, machine learning models require robust computational resources for training and inference, mainly when dealing with extensive datasets of point clouds. To mitigate potential performance bottlenecks, we must investigate high-performance computing infrastructure that seamlessly supports both the rendering tasks and the machine learning processes, ensuring that the overall system operates efficiently and meets project timelines.

Given the nature of our project, an average desktop computer will suffice for our computational needs. The hybrid Gaussian raytracer and the machine learning models can be efficiently run on standard hardware to optimize resource usage. While we will be processing point clouds and rendering complex scenes, the operations involved are manageable within the capabilities of a typical consumer-grade machine. Therefore, we likely won't require high-end computing resources, making the project accessible and cost-effective. This allows us to focus on development without needing specialized or expensive hardware.

4 Design

4.1 Design Exploration

4.1.1 Design Decisions

- We chose to use the Unity engine as the framework for our system. This reduced overhead and built complexity for us in running compute shaders and provided a clear path for the packaging and distribution of our system.

- We chose to use a Raytracer in place of Gaussian Rasterization. Some implementations of Gaussian Splatting use an algorithm called Gaussian Rasterization to render the image; however, these solutions are usually unreliable, limiting our system's applications. Because of this, we decided to use a raytracer, which allows us to change the lighting in the scene and use PBR to render the Gaussians.

- We chose to target 30 FPS. This means that our solution will render scenes in real-time and that our raytracing implementation needs to run efficiently.

4.1.2 Ideation

One of the first choices we made was to use Unity Engine. To make this decision we discussed alternatives such as using C++ and OpenGL or Vulkan to dispatch our compute shaders. These options require a more complex build pipeline and would make much of our implementation platform dependent. Using the Unity engine abstracts many issues and frees us to focus on other system elements.

4.1.3 Decision-Making and Trade-Off

	Pros	**Cons**
Base OpenGL / Vulkan	No existing render pipeline	It is complicated to build and display output. Different platforms will have different compatibility
Unity Engine	Integration with an already existing tool Built-in systems for dispatching Compute Shaders Unity engine provides solutions for cross-platform compatibility	Our system ignores the existing render pipeline

Figure 4: Pros and cons table for using Base OpenGL/Vulkan and The Unity Engine.

We decided to create our project within the Unity engine; this gave us the benefit of not having to set up our window to display the output, implement our update loop, or handle cross-platforming building for low-level C++ code. This decision saved us much time and effort on problems outside our project's core purpose.

The downside of this decision is that Unity has a built-in render pipeline, which our solution doesn't us,e so there is a performance cost to using Unity, but we decided the benefits outweigh the cost.

4.2 Proposed Design

4.2.1 Overview

The Hybrid Relightable 3D Gaussian Rendering System is a real-time novel view synthesis tool that leverages advanced machine learning techniques, 3D geometry processing, and optimized rendering methods to create high-quality, interactive 3D scenes. The system is designed to work with image-based data (such as video or image sequences) and polygonal models, seamlessly integrating them into a unified rendering pipeline.

The process begins by capturing a video or a sequence of images, which are then processed using Structure from Motion (SfM) techniques. This process generates point clouds—collections of 3D points representing the scene's structure. From these point clouds, the system creates Gaussian splats, simplified representations of the scene's geometry, with each point in the cloud representing a Gaussian distribution in 3D space.

Next, the generated Gaussian points are fed into a PyTorch-based optimizer. This optimizer utilizes machine learning techniques to refine the positions and properties of the points. Additionally, it creates extra attributes crucial for accurate rendering, such as material properties, lighting information, and surface details. This optimization ensures the final 3D scene is visually realistic and computationally efficient.

Once the points are optimized, they undergo a material prediction process. A dedicated machine learning model estimates the physical properties of each object in the scene, such as reflectivity, roughness, and transparency. These material properties are critical for realistic lighting and shading in the final render.

The output from the material prediction model is then passed through a parser that prepares the data for rendering. This includes creating a Bounding Volume Hierarchy (BVH), a hierarchical structure that accelerates the ray tracing process by organizing the 3D objects in the scene for more efficient intersection tests. For scenes requiring more detailed geometry, users can integrate polygonal meshes (triangle meshes), which are also processed into their own BVH structures, enabling them to coexist and be rendered alongside the Gaussian splats.

The core of the system's rendering capability lies in its ray tracer. This component takes the BVH data (whether Gaussian or triangle mesh-based). It combines it with the Unity camera and scene lights to calculate how light interacts with the scene, ultimately producing realistic images. The ray tracing process accounts for various optical effects, such as shadows, reflections, and refractions, ensuring the final output is both physically accurate and visually compelling.

The system is packaged as a Unity asset, making it easily deployable for real-time applications. This package provides users with a streamlined interface for importing their own video/image data or polygon meshes, optimizing the scene, and rendering it with real-time feedback.

Through this integrated approach, the Hybrid Relightable 3D Gaussian Rendering System allows users to create complex, physically accurate 3D environments that can be interactively explored and relit, all within a real-time rendering framework.

4.2.2 Detailed Design and Visual(s)

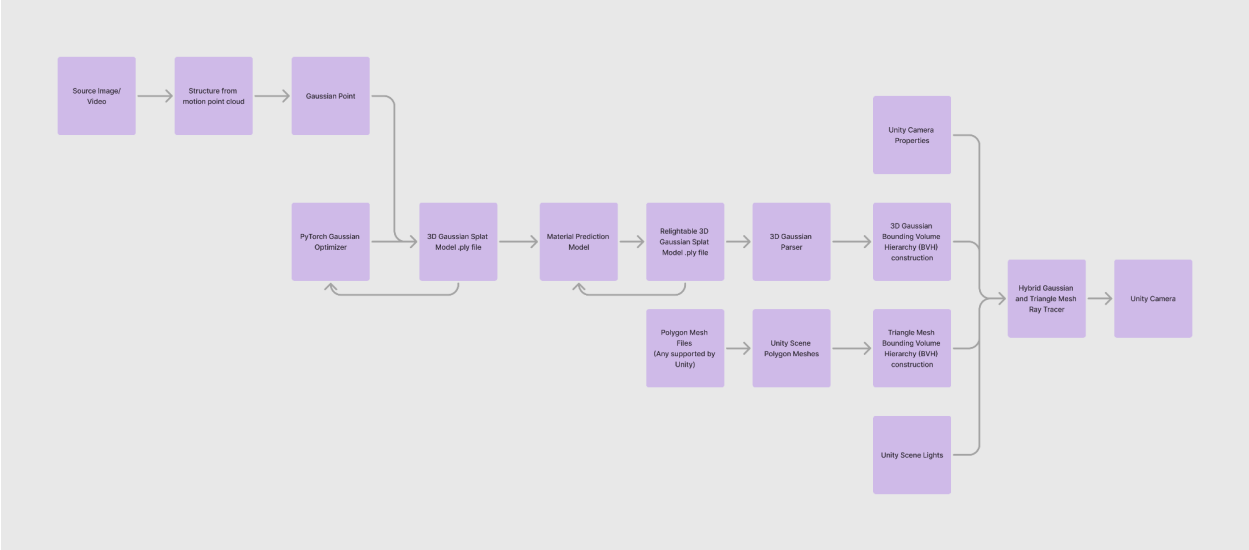

Figure 5: High-level overview of the Hybrid Relightable 3D Gaussian Rendering System.

As the overview explains, our solution consists of a pipeline that takes in images or videos and ultimately outputs a rendered scene. Our system consists of multiple components along this pipeline that modify the data.

1. The first component of the solution is a structure from the motion solution, which creates a cloud of Gaussians from a series of images or videos.
    a. This solution can either be implemented as a part of the project or an existing solution can be used.
2. The second component is a machine learning model that uses the Gaussian cloud and outputs a new One with PBR properties and normal vectors attached.
    a. Precisely, the PBR properties that should be associated with the Gaussians are roughness, metalness, specular, opacity, ambient occlusion, refraction, and emissive; these should all be floats between 0-1, which together describe the material of the Gaussian. In addition, the Gaussians should have an albedo color specified by four floats.
    b. These Gaussians should then be outputted to a file so they can be read in and used by the ray tracer.
3. The final component is the raytracer itself. The raytracer should take in the Gaussian output by the machine learning model and any traditional polygon meshes and render them together in a 3D scene.

a. The raytracer should allow the user to configure the camera's position and lighting by editing the Unity scene or other methods.
b. For performance reasons the raytracer should construct a bounding volume hierarchy to accelerate calculating the ray intersections.
c. The output of the raytracer should be in real-time (30 FPS) and allow the user to update the lighting and camera positions while running.

4.2.3 Functionality

The System is designed to operate seamlessly within a user's workflow, enabling real-time 3D scene reconstruction and rendering. To start, a user will interact with the system through a Unity package, which serves as the interface for input and output. Depending on their needs, the user can upload a video/video-derived image set or a polygon mesh into the Unity package.

Suppose the input is a video or a set of images. In that case, the system will first extract the 3D geometry using Structure from Motion (SfM), generating a point cloud for further optimization. This point cloud will then be processed through a PyTorch-based optimizer, which refines the points and predicts material properties, such as surface reflectance and roughness, based on the visual data. The system then constructs a Bounding Volume Hierarchy (BVH) to organize the scene for efficient ray tracing and rendering. This optimized scene, including material properties and geometry, is ready to be rendered with our custom Unity Render Pipeline, ensuring that the lighting and materials are accurately simulated in real time.

If the user uploads a polygon mesh, the system similarly processes the geometry through BVH construction without the SfM step. The polygon mesh will be integrated into the scene alongside the 3D Gaussian splats, with ray tracing performed on both geometry types for optimal performance.

Once the scene is rendered, the user can interact with the final output directly within Unity or export the rendered scene for use in various applications, such as VR, AR, or alternative engines. The system provides flexibility, allowing users to apply the rendered 3D scenes to their specific use cases, whether for simulations, visualizations, or immersive experiences.

4.2.4 Areas of Concern and Development

At this stage in the development of the Hybrid Relightable 3D Gaussian Rendering System, several areas need to be further explored and refined as part of the ongoing prototyping process. One key concern is ensuring the system can scale effectively for real-time rendering of complex scenes. To ensure our system provides the best performance for the increasing complexity of the system, we will need to continue to monitor and review performance during the prototyping phase. We will need to specifically keep an eye on the BVH and Machine learning prototypes as these two processes will likely be the most significant bottlenecks to performance.

Quality is another central area of concern for our project. It will be necessary to our users that we provide the highest quality scene generation possible. Our design addresses this concern with the inclusion of our material predictor. This material predictor and the values it generates will enhance the quality of the scene.

These additional values allow our renderer to accurately create reflections, shadows, and roughness, all contributing to the realism expected from our user needs.

The final area of concern is ensuring that, once the rendering pipeline is fully developed, it integrates seamlessly into Unity with an intuitive and user-friendly UI. The interface must support users of varying skill levels, from beginners to experienced professionals, allowing them to easily navigate the system's features and efficiently utilize the package. In addition to basic functionality, the UI should provide transparent workflows for scene optimization and rendering, ensuring that users can access advanced features without becoming overwhelmed. Ensuring smooth integration and a positive user experience will be critical to the system's success in real-world applications.

4.3 Technological Considerations

The Hybrid Relightable 3D Gaussian Rendering System integrates various advanced technologies, each chosen for its specific strengths in optimizing 3D rendering, machine learning, and real-time performance. Below, we discuss the key technologies used and their strengths, weaknesses, and trade-offs.

One of the core technologies employed in the system is Unity, a widely used real-time 3D engine. Unity provides a comprehensive platform for developing interactive applications with extensive real-time rendering, physics, and asset management support. Its primary strength lies in its ability to render complex 3D scenes in real-time, making it ideal for applications requiring high interactivity, such as virtual and augmented reality environments. Unity also supports deployment across various platforms, from desktops to mobile devices and VR headsets, ensuring the system's accessibility across diverse hardware. However, the default rendering pipeline in Unity needs to be better suited to the specific requirements of this system. Given the need for custom rendering techniques, such as those for Gaussian splats, the existing pipeline cannot provide the level of control and optimization required. As a result, we must create a custom rendering pipeline derived from Unity's universal rendering pipeline to gain the flexibility needed for these specialized tasks.

Another key technology in the system is PyTorch, an open-source machine learning framework widely used for developing and training deep learning models. PyTorch is utilized in this system to optimize the 3D Gaussian splats, refine point cloud data, and predict material properties based on the scene's geometry. One of the significant strengths of PyTorch is its flexibility; it supports a wide range of machine learning techniques, from supervised learning to reinforcement learning, which makes it highly adaptable to the specific needs of 3D scene reconstruction and material prediction. Additionally, PyTorch's built-in GPU acceleration allows for significant performance improvements, especially when training models or optimizing large datasets in real-time. However, PyTorch is computationally expensive, particularly with large, high-resolution datasets or complex model architectures. Training machine learning models requires substantial computational resources, and optimizing Gaussian splats can introduce delays if not managed carefully.

The technologies chosen for the Hybrid Relightable 3D Gaussian Rendering System represent a balance between flexibility, performance, and real-time rendering capabilities. Unity offers the advantage of a powerful real-time rendering engine with cross-platform deployment, though it requires careful management of low-level rendering and algorithm integration. PyTorch provides flexibility and power for machine learning-based optimization and material prediction, but it also introduces computational overhead that must be managed for real-time use.

4.4 Design Analysis

So far, our progress has come in three major areas: the command buffer for our compute shaders, finding training data for our machine learning models, and designing files and readers to handle taking in data from various steps in our design. One of the earliest concerns we had for the feasibility of our design was whether there was enough available training data to properly create our machine learning model. However, when we started to look for this data, we quickly found a good amount of valuable data, which reassured us of the feasibility of our plans for the machine learning model.

As part of creating files to store our Gaussians to help us test our files, develop a prototype for our system, and demonstrate how our system will work to others, we implemented a shader in Unity, which renders Gaussians on a 2D plane. This prototype works by reading in a 3D Gaussian and passing its information to a shader attached to an aircraft in Unity. We then rendered the Gaussian output on the plane as if its center was within the plane and the plane's color represented a cross-section of the Gaussian(s). While simple, this prototype allows us to see how Gaussians can be used to create renders and images and demonstrate the fundamentals of our system to others.

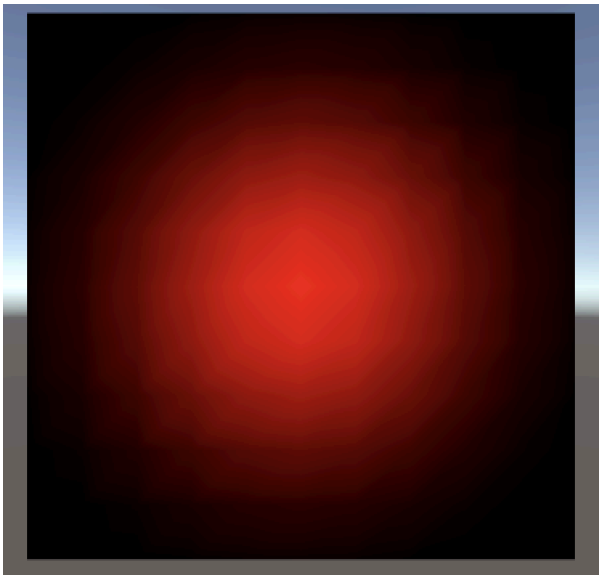

Figure 6: A plane with a single Gaussian centered at its center

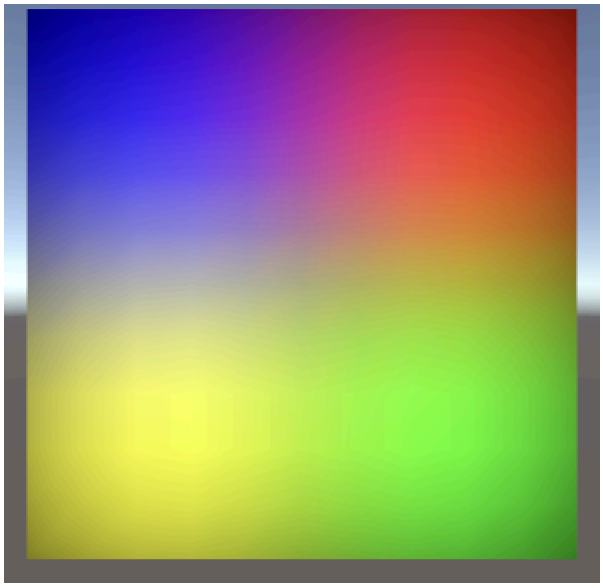

Figure 7: A plane with four overlapping Gaussians placed on it. Some pixels have their color determined by multiple Gaussians.

These prototypes have been incredibly effective at demonstrating the feasibility of our design, as we now have a clear path for rendering images and choosing colors based on Gaussians, which can be recreated in our Raytracer.

5 Testing

Our testing philosophy emphasizes early and iterative testing to ensure all system components meet the design requirements. Given the complexity of our Hybrid Relightable 3D Gaussian Rendering System, the testing must address the real-time rendering performance, machine learning optimization, and seamless integration between subsystems. Described below are the methods and tools used to do this.

5.1 Unit Testing

Our unit testing is taking our baseline image and our new image generated by our unit change to the code. We then will compare the images pixel by pixel and see if they are the same value with a small range of difference to adjust for the “noise” of random light. Some Python libraries can expedite this task or we could always code ourselves.

5.2 Interface Testing

We have the custom Unity rendering pipeline and PyTorch-based optimization models for interfaces. Some interface testing ensures seamless data exchange between these units. One example would be testing whether Unity correctly receives optimized Gaussian parameters from PyTorch and renders them correctly. For tools we can use the Unity Test Framework and PyTest for the python.

5.3 Integration Testing

The critical integration paths: The handoff from PyTorch-optimized data to Unity's rendering engine. The next is the synchronization of Gaussian splats with the scene’s lighting and camera movements. These Integration tests simulate realistic scenarios like dynamically adjusting Gaussian splats in real time during user interactions.

5.4 System Testing

The System-level testing combines unit, interface, and integration tests. An example would be to render a complex scene with multiple Gaussian splats, adjust the parameters via the machine learning model, and then verify the final rendered image for accuracy and performance. Some key requirements tested would be the rendering speed, image quality, and material property predictions. The Unity Editor’s Play Mode Tests and performance benchmarking software are some tools to automate this process.

5.5 Regression Testing

The plan for Regression Testing is to automate it and run it after every System update to ensure new features do not break existing functionality. When making updates, we will prioritize keeping key features such as real-time rendering performance and accurate Gaussian representation above all else.

5.6 Acceptance Testing

We will demonstrate that design requirements are validated by constantly communicating with the clients for this project and the professor overlooking our work. Demonstrations involve running test scenes with predefined requirements, such as accurate material prediction and smooth interactivity. The tools will be feedback forms, requirement checklists, and real-world scene render tests.

5.7 Results

Currently, we have done very minimal testing, but the tests we have done indicate that we are on the right track. All the critical features are working at an adequate level.

6 Implementation

Our team's development was split into two primary groups: the computer graphics and Machine Learning (ML) groups. This was done to efficiently develop the independent core components of our system in

parallel. The graphics team is responsible for developing the hybrid ray tracing rendering solution in the Unity game engine, and the ML team is tasked with generating a point cloud given a series of real-world input images using Structure from Motion (SfM), which will then be used to create a relightable Gaussian model.

During the first semester, the graphics team primarily focused on creating the foundations for our hybrid rendering solution. This consisted of overriding Unity's rendering pipeline with our own custom one. The Unity scripting API allows you to define new command buffers, a series of commands you wish the GPU to execute, during a specific Unity rendering loop phase. The general architecture of our hybrid renderer inserts itself at the end of everything, `CameraEvent.AfterEverything`. There, it overrides the image Unity's universal rendering pipeline generated with a new ray-traced image. To create the ray-traced image, a ray is generated for each pixel, originating from the camera's origin and passing through the center of the pixel. For each generated ray, we determine if it collides with anything in the scene. If a ray collides with an object, it performs a series of lighting calculations depending on material properties, incident direction, and incoming light direction. After this lighting calculation, we reflect this ray off the object's surface and continue tracing it through the scene. Once all rays have reached some arbitrary bounce limit, the image has finished rendering.

Figure 8: A raytraced image of the Stanford Dragon with no scene lighting and a solid color unlit material.

For simplicity, the graphics team focused primarily on triangle mesh integration before Gaussian model integration due to the abundant resources on the former. This allows us to guarantee the accuracy of our physically based lighting calculations before adding in the additional layer of complexity of non-traditional models. The graphics team's end-of-semester one deliverable consisted of a traditional pinhole camera that can render triangle mesh objects with emissive materials, serves as our scene's light source, and Lambertian diffuse materials, the most basic physically based material.

Even with the parallelized nature of performing these calculations via compute shaders, checking for ray intersections for every triangle in the scene is extremely costly. To sidestep this issue, we use bounding volume hierarchies, an industry-standard solution to this issue, to drastically increase the performance of our ray tracer. A ray will only consider triangles that lie within a volume. Thus, we can create a structure of embedded volumes or bounding boxes, where we recursively check for ray collision against these bounding boxes.

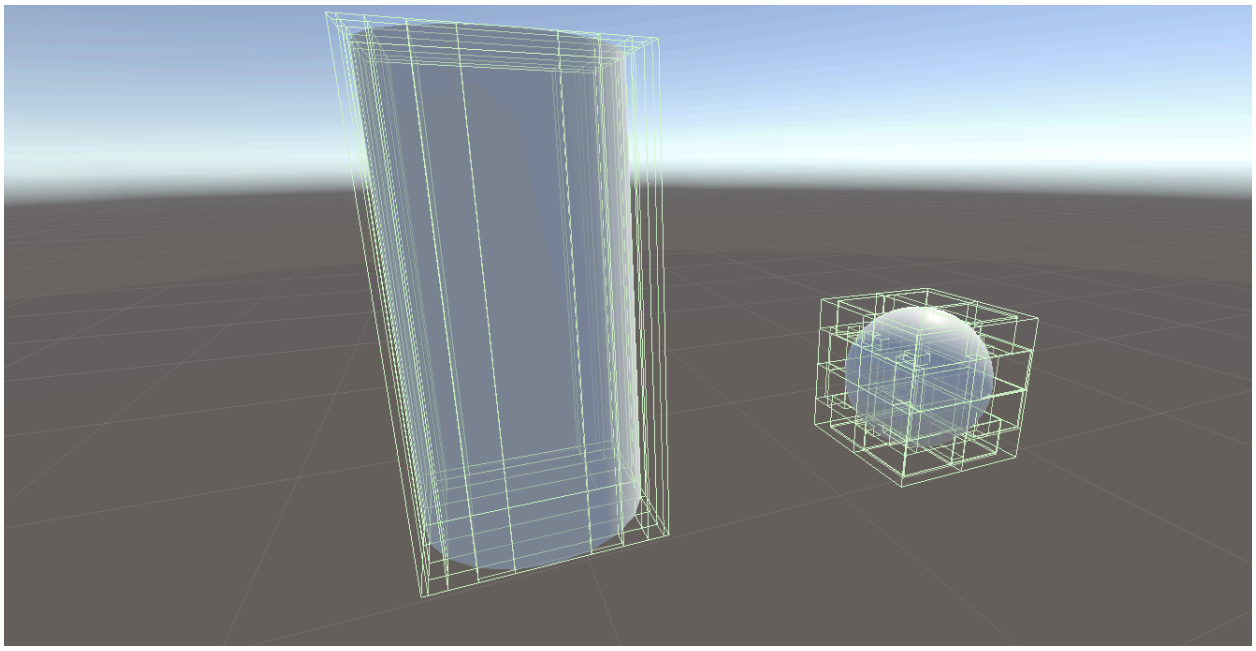

Figure 9: A cylinder and cube with their AABBs visible

The primary focus of the ML team in the first semester was to generate a 3D point cloud from a series of scene images using the Structure-from-Motion (SfM) computer vision method. The team has utilized the open-source program COLMAP and its Python library, PyCOLMAP. We tested our dataset with the COLMAP program to ensure that the results meet our requirements and are compatible with the design of our Gaussian Optimizer. However, for the final implementation, we plan to use the PyCOLMAP library, which provides access to the core functions of COLMAP. This choice is driven by our intention to integrate the functionality as a Unity package, as COLMAP's entire program is incompatible with Unity packages.

Figure 10: A generated point cloud of a train car using the COLMAP. The red boxes represent camera/picture locations.

We implement a Gaussian model and Gaussian optimizer to enhance optimization efficiency in our approach. By default, Gaussians are not relightable, meaning they don't account for changes in lighting or environmental factors. To make them relightable, we input basic Gaussian parameters into a system that derives additional values like reflectivity, transparency, and other material properties that affect light interaction. This enables the Gaussians to adapt to lighting changes, improving the model's realism while maintaining computational efficiency. We plan to use gradient descent optimization algorithms to fine-tune the Gaussian parameters, iteratively adjusting them to minimize errors and improve overall system performance.

Additionally, our team focused development efforts on learning what it will take to place our final product on the Unity Asset Store, Unity's market place to purchase assets and packages. We created a Unity publisher account for our senior design team and created and listed a test package that includes our ray tracer prototype on this account. This was done to understand the process of packaging a Unity project and the steps required to list a package on the Asset Store.

7 Ethics and Professional Responsibility

7.1 Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	**Definition**	**ACM Code of Ethics and Project Application**
Work Competence	Provide work demonstrating excellence, high integrity, punctuality, and professional competence.	2.6 Perform work only in areas of competence. Our project involves advanced topics such as GPU graphics programming, the Unity engine, and

		machine learning models using Pytorch. Ensuring work competence, continuous learning, and practice from all team members is ongoing.
Financial Responsibility	Provide products at a reasonable cost in comparison to the quality of the product available.	**1.3 Be honest and trustworthy** Due to our plan to create a unity package and project circumstances, our product will be free to use.
Communication honesty	Report work understandably and truthfully in such a way as to avoid deception.	**1.3 Be honest and trustworthy** To ensure communication honesty, as a team we meet each Friday to discuss achievements and create a weekly report that is reviewed by each individual to ensure no deceptions are reported.
Health, Safety, Well-being	Minimize risks to individuals' safety, well-being, and health by adhering to safety standards and actively addressing hazards.	**1.2 Avoid Harm** We plan to integrate industry standards like traditional Polygon meshes into our renderer to avoid harm to potential jobs and job states of current 3D artists.
Property Ownership	Respect the property and ideas of others by recognizing their rights and avoiding unauthorized use or infringement.	**1.5 Respect the work required to produce new ideas, inventions, creative works, and computing artifacts.** For our project, we plan to credit all papers, libraries, including the SfM Colmap library, and additional resources used to create our final project.
Sustainability	Protect the environment both on a local and global scale through ethical practices and by avoiding overconsumption.	**1.2 Avoid Harm** We understand that machine learning models and computer graphics can be computationally expensive and require lots of energy, so to alleviate this issue, we plan to include BVH into our renderer, among other optimization techniques, to limit rigorous computations and energy usage.
Social Responsibility	Create a product that contributes positively to society by addressing key needs, improving quality of life, and promoting overall well-being.	**1.1 Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing.** Our system will directly benefit our users and stakeholders by offering a faster, higher quality, more flexible solution to the current industry

		standard for Novel View Synthesis

Figure 11: Areas of responsibility followed by the corresponding definition, then ACM Code of Ethics and Project Application

Our team is performing exceptionally well in the area of work competence. This is crucial in our project, which involves complex and advanced topics like GPU graphics programming, The Unity engine, and machine learning models using PyTorch. We demonstrate excellence by ensuring every team member continuously learns and hones their skills. This commitment to ongoing education, whether through self-study, peer learning, or practical application, allows us to provide high-quality, professional work. Each member is encouraged to work within their areas of competence, which results in timely, accurate, and impactful contributions to the project. Individual areas of competence were determined during our first few team meetings, personal interests, and previous skills/experience. Our dedication to maintaining high integrity and punctuality in delivering tasks ensures that we consistently meet our objectives and deadlines.

While we perform well in work competence, we recognize the need for improvement in communication honesty. We hold weekly meetings to discuss our progress and generate a report each team member reviews to ensure transparency. However, there have been occasions where the clarity and thoroughness of our task documentation could be improved. We plan to implement more rigorous task documentation practices to enhance communication honesty. This would ensure that each task is thoroughly documented with clear progress updates, challenges, and solutions. By doing so, we can avoid any potential misunderstandings or misrepresentations of our work, ensuring that all reports are truthful and understandable. This improvement will strengthen our team's ability to report progress transparently and accurately.

7.2 Four Principles

	Beneficence	**Nonmaleficence**	**Respect for Autonomy**	**Justice**
Public health, safety, and welfare	Our system will directly benefit our users and stakeholders by offering a faster, higher quality, more flexible solution to the current industry standard for Novel View Synthesis	Our system does directly or indirectly harm our stakeholders and users	After creating a minimum viable product, we plan to send out prototypes to stakeholders and receive user feedback before the system is finalized	Our system will be neatly contained in a Unity Package and will be freely available to download on the Unity Asset Store
Global, cultural, and social	Our system is advancing the research of computer graphics and machine learning industries and its	Our system does not go against or disrespect any particular culture or social group	Our system could support cultural practice by digitally capturing one-to-one recreations of culturally	Our system will remain culturally accessible by offering a visual demonstration of how to use our system

	mainstream adoption		significant locations (e.g., religious sites)	
Environmental	Our system captures the beauty of real-world environments, which may inspire users to be more environmentally conscious	Even though our system uses machine learning, our model's training is not computationally expensive when compared to LLMs	Our system does not require the user to re-train the relightable Gaussian model locally	Our system will not affect any environmental habitat
Economic	Our system increases the production quality of renders for smaller companies, therefore creating more competition	We mitigate any potential 3D artist job loss by allowing full support of industry-standard tooling	We will make our product free of charge	Our system will not financially affect our users

Figure 12: Four ethics principle chart and how it applies to our project.

The benefit we are working towards is a faster, higher quality, and more flexible solution for Novel View Synthesis, which will directly enhance user experiences and stakeholder satisfaction. We aim to deliver faster results without compromising quality by leveraging advanced algorithms and optimizing computational efficiency. Additionally, we will incorporate adaptive features that allow users to customize the solution to their specific needs, ensuring greater flexibility. We will continuously test the system with real-world data to ensure success, prioritize user feedback, and refine our models based on iterative improvements.

One broader context-principle pair in which our project may be lacking is the potential impact on 3D artists' job security due to automation and advanced tools. While our system supports industry-standard tooling and enhances efficiency, this could inadvertently raise concerns about job displacement in the 3D artist community.

To overcome this, we focus on the positive aspects of the design, which empower 3D artists rather than replace them. By integrating industry-standard tools, including traditional polygon meshes and our relightable Gaussians, we enable artists to leverage automation to reduce repetitive tasks, allowing them to focus on more creative and complex aspects of their work.

7.3 Virtues

7.3.1 Team Virtues

Collaboration is central to our success, as we rely on diverse expertise and input to create an innovative solution. To foster cooperation, we prioritize open communication and a culture of trust where each team member feels comfortable sharing ideas and feedback. We hold weekly team meetings and biweekly meetings with our advisor, Dr Mitra to discuss progress, challenges, and brainstorming sessions to ensure

everyone is aligned on goals. Additionally, we use collaborative tools and platforms, such as Google Drive (docs, slides, etc) and Figma, that allow for real-time sharing and problem-solving, promoting collective ownership of the project.

Innovation is another central virtue of our team. Innovation is the driving force for our approach to solving Novel View Synthesis problems. We encourage innovation by creating a team environment where experimentation is encouraged, and each team member can propose and test new ideas. We also promote innovation by continuing to explore new algorithms, techniques, libraries, and software to continue improving our project's performance and quality. We also prioritize learning and staying up to date with the latest papers and research, with some being released during our design process. All in all, innovation ensures that our team has the knowledge and resources to innovate effectively.

Empathy is an essential portion of our virtues as empathy allows us to understand our users' needs and potential areas of concern. As previously mentioned, we aim to create a unity package that assists in the creative process for 3D artists by including our AI tools rather than outright replacing the work they are assigned. We plan to continue addressing this virtue by gathering and analyzing user feedback to ensure our solution meets their needs and genuinely assists and benefits their work. We are mindful of how our package will affect the user's workflow and will continue to empathize to ensure the best product we can provide.

7.3.2 Jackson Vanderheyden's Virtues

One virtue I have demonstrated in our senior design work thus far is diligence. This virtue is important to me because it shows a strong commitment to the project's success despite unexpected hardships. I have demonstrated this virtue by consistently contributing significant time towards our project throughout the semester. I created an educational resource for the team and developed a stable foundation for our hybrid rendering solution. Additionally, I made myself available to support team members, which fostered collaboration and progress. This has helped the project by ensuring the ease of future development for the graphics portion of this project and by providing resources and assistance to team members.

A virtue that is important to me but I have not demonstrated as fully in my senior design work so far is attentiveness. This virtue is important to me because it ensures that all project aspects are fully addressed and all members are aligned with their goals. I have not demonstrated this virtue fully because I did not consistently hold our team accountable for the soft deadlines we set for ourselves and occasionally prioritized other classes over ensuring the project was progressing as it should. There also needed to be more communication between the graphics and machine learning teams, leading to inefficiencies and misalignments, which could have been addressed proactively. To demonstrate this virtue more effectively, I plan to take a more active role in team management by regularly checking in with all team members and setting more explicit weekly goals to ensure alignment across the team.

7.3.3 Brian Xicon's Virtues

One virtue I have demonstrated in our senior design work thus far is responsibility. This virtue is important because it shows that I can be trusted to complete tasks important to our team's success. I have demonstrated this virtue by meeting my deadlines, and ensuring the machine learning modules are thoroughly made. This has helped the project by maintaining steady progress in our machine-learning tasks.

A virtue that is important to me but I have not demonstrated as fully in my senior design work so far is foresight. This virtue is important to me because thinking ahead and anticipating potential problems allows

our team to allocate the necessary time to address them. I recognize that I have not demonstrated this virtue fully because sometimes I focus on immediate tasks and have difficulty giving myself adequate time for certain tasks. To demonstrate this virtue more effectively, I plan to dedicate more time during our sprints to identify possible upcoming challenges to give myself adequate time to solve them.

7.3.4 Luke Broglio's Virtues

One virtue I have demonstrated in our senior design work thus far is persistence. This virtue is important to me because I think it is essential to the success of any project and is a strength I value. I have demonstrated this virtue by continuing to work on our project even when I encounter complicated or hard to diagnose issues. This has helped the project by allowing me to finish demos or project components quicker.

A virtue that is important to me but I have not demonstrated as fully in my senior design work so far is collaboration. This virtue is important to me because it helps increase work speed, quality, and is important for integrating different components. I recognize that I have not demonstrated this virtue fully because I haven't reached out to my other team members about my work until it was finished or been available to help others as often as I should. To demonstrate this virtue more effectively, I plan to send my group regular updates about my work and make sure others know when I am available to help.

7.3.5 Ethan Gasner's Virtues

One virtue I have demonstrated in our senior design work thus far is accountability. This virtue is important to me because I believe if people are held accountable for their actions it results in a more productive and efficient work environment. I have demonstrated this virtue by ensuring that all documentation assignments and weekly reports are finished and turned in on time. I specifically check on the progress of assignments and notify individuals as a reminder to finish their assigned portion of these documents. I am also the individual responsible for turning in these documents and updating the team website as well. This has helped the project stay on track and ensure documentation is finished on time.

A virtue that is important to me but I have not demonstrated as fully in my senior design work so far is software sustainability. This virtue is important to me because I would like to create software that is not only impactful but will be used for a long period of time. I have not demonstrated this virtue fully because, at this point in our project, most of our work has only been prototypes and basic implementations without much concern for software sustainability. To demonstrate this virtue more effectively, I plan to structure the code in such a way that the code can easily be updated and improved upon when needed by including adequate comments and code and documentation on code functionality.

7.3.6 Kyle Kohl's Virtues

One virtue I have demonstrated in our senior design work thus far is enthusiasm. This virtue is important to me because I believe that without it, the work becomes a grind and motivation to keep working disappearers. I have done my best to demonstrate this virtue by encouraging and helping out my team however I can. I can't say I have done this perfectly, but I believe I have done my best. Asking if there was something I could help with to further the project by solving a teammate's problem and discovering the problem was with his computer.

A virtue that is important to me but I have not demonstrated as fully in my senior design work so far is responsibility. This virtue is important to me because responsibility is the basis for doing a good job. I recognize that I have not demonstrated this virtue fully because I put the soft deadlines of this project in

lower priority than my harder deadlines in other classes. I plan to prioritize this class first next semester to demonstrate this virtue more effectively. I set up my senior year schedule, so the fall semester would be my hard one. I expect next semester to be able to give this class the attention it deserves.

8 Closing Material

8.1 Conclusion

Our project is to create a system for performing novel view synthesis using relightable Gaussian splatting and rendering traditional polygon models. This system will take in a series of images or a video of an object or location in the real world, process them to create a 3D representation of the object or place using structure from motion, convert that output into a detailed scene made up of 3D Gaussians, extract the lighting and Physically based rendering property data from the gaussians, and then render the gaussian scene along with any polygon meshes the user adds to it.

We have four main goals for this system: the output it creates should be of high quality and updated in real-time, the user should be able to change and control the lighting in the scene, both the Gaussians and polygon meshes should be rendered using Physically based rendering, and the final solution should be published as a Unity package,

To help us make a plan to achieve these goals we broke our system into its two primary components.

First, we have the AI models, which create the 3D Gaussian scene from the processed input images and extract the lighting and PBR data for the Gaussians. One of the models' functions will be to increase the number of Gaussians in the scene. This will be accomplished using machine learning, where we will train a model to extrapolate new Gaussians from the ones created during the structure from the motion process and increase the density of the Gaussians making up the scene. Increasing the number of Gaussians will significantly improve the final output quality and lead to more detailed scenes. The AI models will also extract the values needed for reliability and physically based rendering. The models will modify the Gaussians in the scene to remove the lighting effects and then extract the normal vectors and PBR properties for each Gaussian so they can be passed to the raytracer.

The second component is the raytracer, which renders the scenes once the Gaussians have been created and processed. To allow for real-time rendering, the raytracer will be implemented on the GPU using compute shaders to do the calculations in parallel. To further speed up the rendering process the raytracer will use a bounding volume hierarchy to speed up the process of calculating ray intersections. The logic for calculating how the PBR properties and user-configured lighting affect the Gaussians and meshes will be contained within the raytracer.

So far, to create this system, we have set files that can be used to store 3D Gaussians and programs to read those files into Unity so they can be given to the raytracer. We have also created a command buffer that will function as the render pipeline for our solution, and within that, we have implemented the raytracer for polygon meshes. We have started to train our models and successfully created models that extract several of the needed PBR properties. Finally we have started researching how to package our design for publishing as a Unity package and worked on publishing smaller test packages.

Our most significant challenges in creating this system so far have come from working around restrictions imposed by the tools and technologies we are using. Specifically, we had to work around the limits of running on the GPU and trouble packaging some elements of our solution, such as trained machine

learning models, into a unity package. To overcome these challenges in the future, we hope to gain more experience with GPU programming to help us learn what techniques and methods work best and to learn how similar projects package themselves within Unity.

8.2 References

B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering," ACM Transactions on Graphics, vol. 42, no. 4, Jul. 4AD, [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

Gao, J., et al. "Relightable 3D Gaussian: Real-time Point Cloud Relighting with BRDF Decomposition and Ray Tracing," in arXiv:2311.16043, 2023.

Nicolas Moenne-Loccoz, undefined., et al. "3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes," in ACM Transactions on Graphics and SIGGRAPH Asia, 2024.

8.3 Glossary

Term	Definition
Novel View Synthesis	The process of creating 3D renders or models from a video or a series of photos of an actual world location.
3D Render	The viewable image or scene output by a computer graphics program.
3D Model	A file that contains data describing a (typically) 3D object or scene. These files are generally used as the input of a computer graphics program.
Neural Radiance Fields (NeRFs)	A technology used to perform Novel View Synthesis, which uses a deep learning model to transform a set of 2D images into a 3D scene represented by a radiance field
3D Gaussian Splatting	A technique for performing novel view synthesis works by transforming the images into a cloud of 3D Gaussians and then rendering it.
3D Gaussian	A 3D space filled with points that follow a Gaussian distribution.
Polygon Mesh	A traditional way of storing 3D models is where the shape of an object's (or collection of objects) surface is defined by vertices, edges, and faces.
Global Illumination	A type of 3D graphics program where indirect lighting (light reflected off other objects) is

Term	Definition
	considered when rendering the scene, not only the light that comes directly from a light source.
Raytracing	A global illumination technique where a 3D scene is rendered by calculating the path of light rays cast into the scene.
Structure from Motion (SfM)	The process of generating the 3D structure of a scene from a set of 2D images or video.
Physically Based Rendering	A way of modeling the real world properties of light and surfaces for the purpose of computer graphics.
Axis Aligned Bounding Box (AABB)	A rectangular volume in a 3D scene which is represented by a minimum and maximum point. They are computationally efficient to calculate intersections with.
Bounding Volume Hierarchy (BVH)	A data structure used to accelerate the calculation of raytracer intersections. It is constructed as a binary tree where each node is an AABB.
Bidirectional Reflectance Distribution Function (BRDF)	A function which determines how light reflects off of a surface.
Compute Shader	A shader program dispatched to the GPU which performs arbitrary computation (instead of being a part of the graphics pipeline).
Frames per second (FPS)	How many times per second an update is rendered to the screen.
Real time rendering	For the purposes of this project, real-time must be at least 30 FPS

Figure 13: Common terms used throughout our design document.

9 Team

9.1 Team Members

- Jackson Vanderheyden
- Brian Xicon
- Luke Broglio
- Ethan Gasner
- Kyle Kohl

9.2 Required Skill Sets for Your Project

- **Machine Learning & Optimization:** This is required to develop our Gaussian optimizer and material prediction models.
- **3D Graphics Programming:** This is necessary to implement ray tracing, BVH, and integration with Unity.
- **Data Processing:** This is required for preparing and handling datasets for our machine learning models.
- **Python & C++:** These programming languages are the main languages we use for our project. Python is used for the machine learning aspect of our project, while C++ is used for the graphics programming aspect of our project.

9.3 Skill Sets Covered by the Team

- **Machine Learning & Optimization Expertise:** Brian Xicon, Ethan Gasner
- **3D Graphics Programming Expertise:** Jackson Vanderheyden, Luke Broglio
- **Data Processing Expertise:** Brian Xicon, Ethan Gasner
- **Python & C++ Expertise:** Brian Xicon, Ethan Gasner, Jackson Vanderheyden, Luke Broglio, Kyle Kohl

9.4 Project Management Style Adopted by the Team

Our team is following an **Agile workflow** with two-week sprints. This allows us to use progressive elaboration to develop our project plan as we gain deeper knowledge of the technologies and techniques used in Gaussian Splatting. This allows us to have flexibility and adaptability in our project process. At the end of each sprint, we meet with our advisor, Dr. Mitra, to showcase our progress through a demonstration to receive feedback.

9.5 Initial Project Management Roles

The following roles serve as general, not rigid, guidelines, and therefore, work traditionally specified for a particular managerial role will not all be exclusively designated to that individual.

- **Graphics Scope Manager:** Jackson Vanderheyden
  - Oversees the graphical elements of the project. This includes ray tracing, BVH, and Unity integration.
- **Machine Learning Scope Manager:** Brian Xicon
  - Leads the development and optimization of the machine learning models for the Gaussian Point Optimizer and material property predictor.
- **Schedule Manager:** Luke Broglio
  - Maintains our project timeline and tracks progress to keep the project on schedule
  - Schedules meeting times and locations.
- **Documentation Manager:** Ethan Gasner
  - Manages all of our project documentation and ensures all our deliverables are well-documented and accessible through our team website.
- **Communication Manager:** Kyle Kohl
  - Serves as the main point of contact for all communication aspects.
  - Acts as the main record keeper for all meetings.

9.6 Team Contract

9.6.1 Team Procedures

1. Face-to-face Friday at 9:30 am serves as our regular team meeting schedule. Additionally, meet on Tuesdays at 4:00 pm bi-weekly with our advisor: Dr. Mitra.
2. Discord will be used as our preferred method of communication for updates, reminders, issues, and scheduling.
3. Consensus-based decision-making policy will be employed, ensuring all members contribute to the discussion.
4. The Communication Manager will take primary responsibility for meeting record keeping. All records will be stored on the relevant Figma page.

9.6.2 Participation Expectations

1. All team members must punctually attend (i.e., be no more than 10 minutes tardy) and participate in all meetings. All absences must be excused 24 hours prior to the set meeting time.
2. All team members are expected to complete their assigned work prior to the agreed upon deadline.
3. All team members must continuously and accurately report progress on the Kanban board and in meetings. Unexpected delays resulting in timeline changes must be reported as soon as possible.
4. All team members are expected to provide feedback on all team decisions and commit the necessary number of hours outside of class time to complete assigned work before the agreed upon deadline.

9.6.3 Leadership

1. Discord will be used as an informal Q&A forum to help support and guide the work of all team members
2. Team members should provide positive peer-to-peer feedback to recognize the contributions of all team members.

9.6.4 Collaboration and Inclusion

1. The skills, expertise, and unique perspectives Jackson Vanderheyden brings to the team are: computer graphics industry experience, multiple Unity projects, & previous experience implementing ray tracing.
2. The skills, expertise, and unique perspectives Brian Xicon brings to the team are: C, C+ +,HTML, CSS, JS, Python, PyTorch experience, Machine Learning experience, and Computer Vision experience.
3. The skills, expertise, and unique perspectives Ethan Gasner brings to the team are: C, C++, python, AI Experience, Machine learning experience, Unity Experience, unique Cyber security perspective, JavaScript, HTML/CSS. Additionally, a cooperative and easygoing attitude.
4. The skills, expertise, and unique perspectives Kyle Kohl brings to the team are: C, C++, Java, Python, a little bit of HTML and CSS. He has the definite advantage of being an extrovert that loves to encourage others. Lots of experience in the communication role.
5. The skills, expertise, and unique perspectives Luke Broglio brings to the team are: Experience with C, C++, python, graphics programming, Unity, HTML/CSS, Javascript, writing a raytracer and experience with working in agile/scrum development environments.
6. We will encourage and support contributions and ideas from all teams through the following: create a safe and open environment, use of structured discussion, leverage team diversity, ask open ended questions, and Active listening.

7. Our group will be very open with each other, if any of us have any concerns with the environment all of us will be open to feedback and work together to discover steps to resolve the issue.

9.6.5 Goal-Setting, Planning, and Execution

1. Our team goal for this semester is to have a working prototype raytracer that uses 3D Gaussians.
2. Our strategy for planning and assigning individual and team work is to assign tasks primarily based on individual role then assign tasks based on interest and best fit.
3. Our strategy for keeping on task are general time management techniques based on the individual.

9.6.6 Consequences for Not Adhering to Team Contract

1. If an individual does not adhere to the above standards, We will start off with verbal warnings from the team and potentially a meeting if necessary. If this continues to become an issue, we will bring this up with the professors and course coordinator.