HyperNeRFGAN

1. Slide 1:

a) Part 1:

We are living in an era of explosive growth in **generative AI**. Almost everyone is familiar with stunning 2D generative models like MidJourney or StyleGAN. However, they have a major limitation.

- Look at the left side of this picture. Traditional 2D models produce very sharp images, but they are flat. Like a picture on a wall, you can't rotate it to see the back or the side.

⇒ But our real world is a 3-dimensional space. Therefore, the modern research trend is 3D-aware Image Synthesis - a combination of the creative capabilities of GANs and the geometric understanding of NeRFs.

- On the right is our destination: A model that not only generates an image of the cat, but also understands the cat's 3D form (Volumetric Understanding). We can rotate and change the viewing angle while maintaining image consistency.

b) Part 2:

It sounds very promising, but why isn't this technology widespread? There are two huge pain points that both the original paper and reality face:

- **First, Computational Cost:** NeRF operates by firing millions of light rays (ray marching) to create images. This is extremely expensive. Most current research (like StyleNeRF or π-GAN) requires powerful GPUs like the NVIDIA A100 or V100 with massive memory. This makes it very difficult for students or independent researchers to access.
- **Second, Data (Pose Dependency):** Traditional NeRF is like a fussy child; it needs to know exactly where the camera is placed (x, y, z coordinates, rotation angle) before it can learn. But in reality, the image data we collect online (e.g., car photos, faces) often lack this information (unposed data)."

c) Part 3:

Therefore, our group chose to research the HyperNeRFGAN paper. This architecture is designed to solve both of the above problems using a **hypernetwork**. However, our group didn't stop at just reading and understanding the theory. Our core goal for this project is:

- To demonstrate that we can reproduce and optimize this complex model so that it runs smoothly on a common, free graphics card available on Google Colab – a Tesla T4 with 16GB of VRAM

---

2. Slide 2:

a) Part 1: Input:

First, let's look at the input data. We have a set of 2D images (such as images of cars or airplanes). The key idea here is the term **'Unknown poses'**. That means, for each image, we have no idea where the camera is placed or the angle of the shot. This is an unsupervised learning problem, which is quite challenging.

b) Objective & Image Generation Mechanism:

Our goal is to build an image generation machine $\hat{I}$, described by this central formula:"

(Pointing to the formula $\hat{I} = \pi(V(z, \theta), \xi)$)

"Let me explain how it works, similar to the process of taking a photograph in the real world:

- First, we have `z`: This is a random noise vector, which you can think of as the 'DNA' that defines the shape of the object (for example: red car, sedan style).
- From this DNA, the model generates `V`: This is the 3D representation of the object (a 3D block).
- Next, we need **ξ**: This is the randomly selected camera position.
- Finally, we have **π**: This is the **Render** operator. It acts like the camera, capturing the 3D block V from the perspective ξ to produce the 2D image $\hat{I}$."

c) Adversarial Game:

But how does the model know if it is generating the correct object or not, without a sample answer? We use the classic **Min-Max Game** mechanism of GANs:"

(Pointing to the final formula)

This is a battle between two sides:

- The **Generator (G)** tries to create fake images ($\hat{I}$) that look as real as possible to deceive the opponent.
- The **Discriminator (D)** acts as the judge, trying to distinguish real images from the dataset (`x`) or which are the fake images generated by the model.

Through this adversarial process, the Generator is forced to learn how to create consistent and realistic 3D objects in order to win.

Slide 3:

(Switch slide and point to the general diagram)

a) Overview:

Next, let me dive deeper into the specific architecture. The diagram on the slide illustrates the flow of HyperNeRFGAN. It consists of three main components, working together in harmony."

b) Component 1: Generator - The Hypernetwork :

"First and foremost is the **Hypernetwork (`G`)**.

Normally, a GAN Generator generates an image (pixel points). But here, it's completely different. This Generator uses the powerful **StyleGAN2** architecture, but its task is to **generate weights (θ)**.

You can think of it this way: this Generator is like a *programmer*. Each time it receives the signal `z` , it writes a *code* (a set of weights) specifically for a new object."

c) Component 2: Target Network (F_θ) - The NeRF:

"So, what are those weights used for? They are fed into the second component: the **Target Network**.

This is the NeRF network that represents the 3D object (e.g., a car). This network receives the weights from `G` and transforms them into a specific car.

In other words: the Hypernetwork (`G`) generates the Target Network (F_θ)."

d) Component 3: Discriminator (D) - The Critic:

"Finally, we have the **Discriminator**. This is a standard component, a 2D CNN network. After NeRF renders the car image, the Discriminator examines that image to assess how closely it resembles a real image."

e) Data Flow Summary (Visual Walkthrough):

(Move hand to draw a path from left to right on the illustration)

"To sum up, the process flows as follows:

From noise `z` → Generator creates Weights → Feed into NeRF → NeRF renders a 2D image → Discriminator evaluates the image.

This entire system is trained End-to-End."

Slide 4:

(Pointing to the first formula - Standard NeRF)

a) Standard NeRF (Old):

"Now, let's dive into the Target Network.

In the standard NeRF architecture (as shown in the formula above), the function F takes in two parameters:

- `x`: The spatial coordinates (x, y, z).
- `d`: The viewing direction. (theta and...)

Including `d` allows NeRF to model complex light effects like reflections on mirrors or metal (where you see bright from one angle, dark from another). However, this makes the input space **5-dimensional**."

(Pointing to the second formula - HyperNeRFGAN)

b) HyperNeRFGAN Improvement (New):

In this paper, the authors apply a bold simplification as shown in the formula below: **Completely remove the viewing direction** `d` **.**

Now, the function F_θ only depends on the coordinates $\mathbf{x}$.

This means that the color of the object at a point is fixed, unchanged no matter which angle you view it from.

c) Conclusion (Key Takeaway):

Mathematically, we've reduced the problem space from **5D to 3D**.

This reduction significantly *reduces the computational burden on the neural network*, paving the way for the model to converge faster on limited hardware."

Slide 5:

a) Definition of Lambertian Assumption (The Physics):

Removing the viewing direction `d` is actually based on a physical principle called the **Lambertian surface assumption**.

Simply put, we treat all objects in the dataset as having perfect matte surfaces, reflecting light uniformly in all directions.

As a result, the observed color is **isotropic**.

⇒ Mathematically, this shows that the variable `d` becomes **redundant**.

b) Why does this assumption save the model? (The "Why"):

This is the most important part. This assumption addresses two critical problems when training on unposed (unlabeled) data:

- **First, it Removes Ambiguity:**

  When the model doesn't know where the camera is, it's very easy to confuse **texture** with **reflected light**.

  For example: A bright streak on a car. Without this assumption, the model might mistakenly think the streak is the car's white paint.

  ⇒ it is because the Lambertian assumption forces the model to understand that it's the fixed color of the object.

- **Second, it Stabilizes Training:**

  By blocking the 'cheating' path (creating optical illusions through the viewpoint), the Hypernetwork is forced to focus entirely on learning **Geometry** and **Texture** accurately.

This is the key to helping the model converge without needing camera information.

Slide 6:

a) The Challenge:

"As I've introduced, the Hypernetwork has to generate weights for the sub-network. But this poses a massive memory challenge.

If the sub-network has layers of size $128 × 128$, the Hypernetwork has to generate a matrix of size 128^2 (over 16 thousand parameters) for each layer.

The complexity is `O(n^2)`. If we followed this naive approach, the GPU memory would overflow immediately."

(Pointing to the FMM formula)

b) The FMM Solution:

To address this, we use a technique called **Factorized Multiplicative Modulation (FMM)**.

The main idea is: Instead of generating the large weight matrix directly, the Hypernetwork generates two 'ultra-thin' low-rank matrices `A` and `B`.

- Matrix `A` is of size `n x k`.
- Matrix `B` is of size `k x n`.

  Where `k` is very small (only about 10).

c) Efficiency Derivation:

"When we multiply `A` and `B` together, we approximate the large original matrix.

In terms of computation, this technique reduces the complexity from quadratic `O(n^2)` to linear `O(n)`.

Specifically, for our configuration, this technique reduces the number of parameters to generate by **6.4 times**.

This is the 'golden key' that enables a complex architecture like HyperNeRFGAN to fit into our modest Tesla T4 GPU."

Slide 7:

a) Integral & Riemann Sum (The Process):

"After having the NeRF network, how do we generate the 2D image? We use the technique of **Volumetric Rendering**.

In theory, the color of a pixel is the result of a continuous integral along the ray (first formula).

However, computers cannot compute an infinite integral. Therefore, we approximate it using a **Riemann Sum** (second formula).

This means that we divide the ray into N_s small segments and sample at those points."

(Pointing to the Mathematical Insight)

b) Why fewer samples cause errors? (The Insight):

"Here, the team wants to emphasize an important mathematical analysis that explains the failure in the upcoming experiments.

In the Riemann formula, the opacity of the object is calculated by the term $(1 - e^{-\sigma \delta_i})$.

Where δ_i is the distance between the sample points.

When we reduce the number of samples N_s too low (for example, 8):

- The distance δ_i between the sample points becomes too large.
- The accumulated error increases drastically.
- As a result, the total opacity never reaches a value of 1.0 (i.e., a solid object). Instead, it creates semi-transparent objects."

c) Conclusion:

"This is the mathematical reason behind the **'Cloudy Artifacts'** (cloudy, translucent objects) that we will visually demonstrate in the results section."

Slide 8:

a) The Problem with GANs:

"Finally, in the methods section, we must address the toughest problem in GANs: instability.

Because our Generator is a Hypernetwork (which generates weights for another network), it is extremely sensitive. If the feedback signal from the Discriminator is too strong or too noisy, the entire model will collapse."

(Pointing to the formula)

b) The R1 Regularization Solution:

To mitigate this, we use the **R1 Gradient Penalty** technique.

Looking at the formula, you can see that we penalize the square of the gradient of the Discriminator ($\nabla D(x)$).

Simply put, we do not allow the Discriminator to change its decision too abruptly when the input changes slightly.

c) Why is it Necessary? (Why Necessary):

This provides three critical benefits:

1. Ensures **Lipschitz Continuity**: Makes the loss function smoother.
2. Prevents **Exploding Gradients**: Keeps the training signal within safe bounds.
3. Creates a 'flatter' optimization landscape, allowing the Hypernetwork to probe and learn gradually without 'slipping'.

Slide 9:

a) The "Memory Wall" Challenge (The Constraint):

The theory sounds great, but when we tried to run the model in practice, we hit a big wall: **VRAM**.

The original model requires GPUs like the A100/V100 with over 32GB of memory.

However, on Google Colab, we only have a Tesla T4 with 16GB. If we run the default config, the model crashes immediately due to out-of-memory (OOM) errors."

(Pointing to the comparison table)

b) The Optimization Solution:

To 'fit' this elephant into a refrigerator, we performed some 'surgery' based on the theoretical knowledge we gathered:

- **First, Reduce Patch Size:** We reduced the render area size from 64×64 to 32×32.

  Note: While the side length is halved, the area is reduced by **4 times**. This is the most memory-efficient factor.

- **Second, Reduce the Number of Samples:** We reduced the number of sample points along each ray from 64 to 32 ($N_s=32$). This is the minimum number of samples to maintain the 3D shape (as we analyzed in the previous slide).

- **Third, Environment:** We used Micromamba and pinned older library versions to ensure compatibility with the hardware.

c) The Result (The Outcome):

Look at the last line of the table.

The total number of points to compute in one pass (Volumetric Load) decreased from 262 thousand points to just 32 thousand points.

This means our model is **8 times lighter** than the original.

As a result, HyperNeRFGAN ran smoothly and converged well on a single Tesla T4 card.

Slide 10: Pre-trained Results

- **Lời dẫn:** "First, we tested on pre-trained weight sets to ensure the code works correctly. The model performed well on four datasets: CARLA, Cars, Chairs, and Airplanes."

Slide 11: Qualitative Baseline Results

- **Lời dẫn:** "Here are the results from our own re-training (Reproduction).

  Despite hardware limitations, the model still generates objects with clear shapes.

  In particular, the Airplane dataset was the most challenging due to the thin wings, but the model was still able to capture the structure."

Slide 12: Manifold Continuity

- **Lời dẫn:** "To prove that the model truly understands the 3D space and isn't just memorizing, we performed **Linear Interpolation**.

  [Pointing to slide] As you can see, when transitioning from Car A to Car B, the shape changes smoothly, and the thickness of the chair legs changes gradually. There is no sign of the image breaking or flickering."

Slide 13: Quantitative Analysis

- **Lời dẫn:** "In terms of quantitative results, the FID score drops significantly after 200K training steps.

  Notably, the ShapeNet Cars dataset improves by 60%. This confirms that the group's memory optimization strategy is effective."

PHẦN 5: ABLATION STUDY & KẾT LUẬN (5 Phút)

(Deep analysis section - "Score points")

Slide 14: Ablation - Geometry-Memory Trade-off

- **Lời dẫn:** "We conducted ablation experiments to find the model's limits.
- **Experiment 1 - Sparse Sampling:** When we reduced the number of samples to 8 (on the left), the object became a blurry 'ghost'. The main reason is the integral error, as explained in Slide 8.
- **Experiment 2 - Small Patch:** When we reduced the patch size to 8×8 (on the right), the car broke apart. This is because the Discriminator could only see local texture spots, losing the global shape context."

Slide 15: Ablation - Stability & Mode Collapse

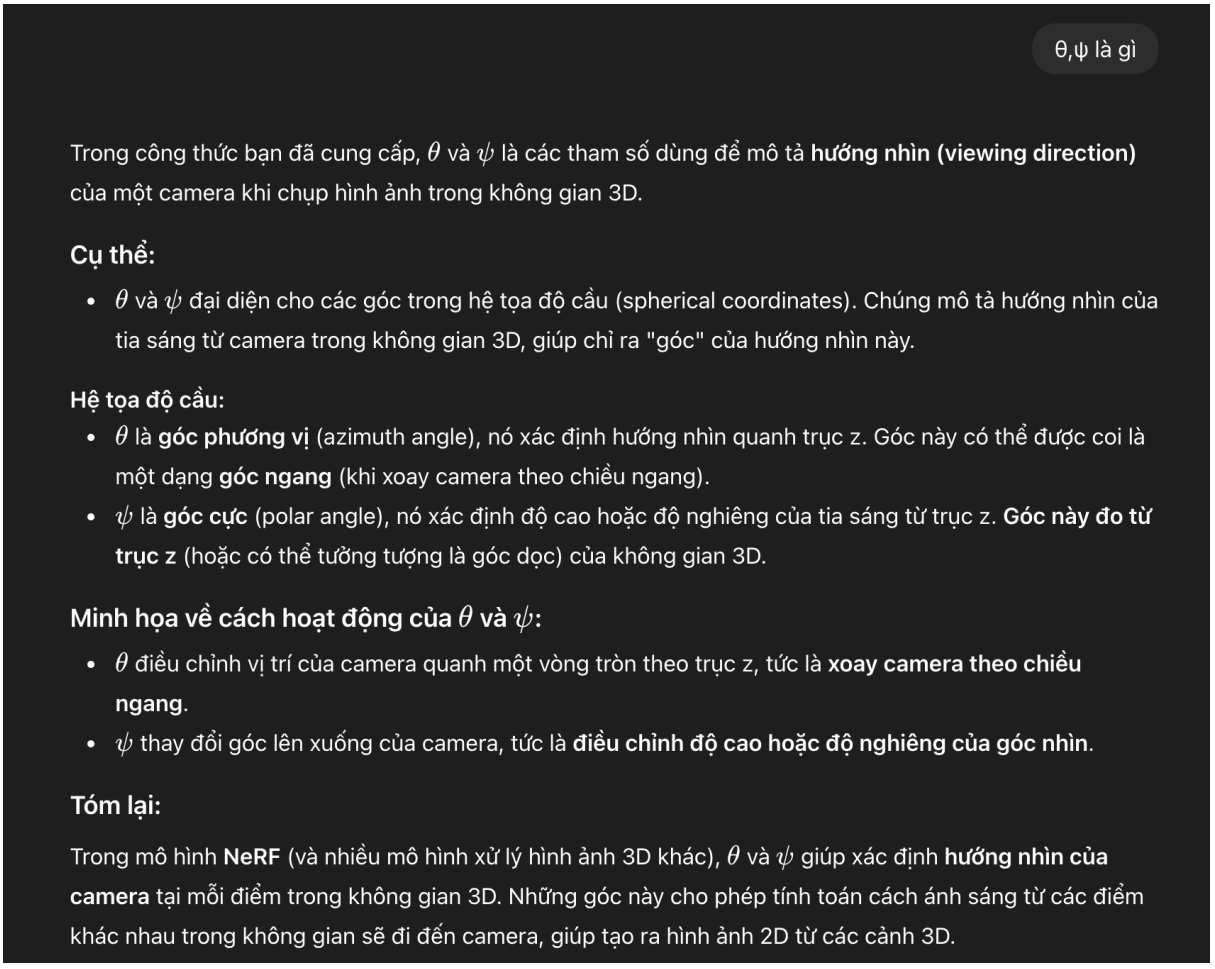
- **Lời dẫn:**
- **Experiment 3 - No Augmentation:** This was an interesting paradox. The FID score was very good, but visually we observed **Mode Collapse** — all the cars looked identical. The Discriminator had overfitted due to limited data.
- **Experiment 4 - Reduced Regularization:** When we reduced the R1 Regularization, the generated images were noisy (artifacts), proving the instability in training."

Slide 16: Conclusion

- **Lời dẫn:** "In summary, the team draws three main conclusions:
  1. **Memory Wall:** There is a hard mathematical limit. We cannot reduce the number of samples below 32 if we want to preserve a solid 3D illusion.
  2. **View-Independence:** This is a fantastic feature that allows the model to work well on unposed data, but it comes with the trade-off of losing specular effects.
  3. **Feasibility:** HyperNeRFGAN can run on standard hardware if tuned correctly.

Slide 17: Thank You

- **Lời dẫn:** "That concludes our presentation. Thank you for your attention, and we welcome any questions and feedback."



1. Tích phân & Tổng Riemann (The Process):

"Sau khi có mạng NeRF, làm sao để tạo ra ảnh 2D? Chúng em dùng kỹ thuật **Volumetric Rendering**.

Về lý thuyết, màu sắc của một điểm ảnh là kết quả của một tích phân liên tục dọc theo tia sáng (công thức đầu tiên).

Tuy nhiên, máy tính không thể tính tích phân vô hạn. Do đó, chúng em phải xấp xỉ nó bằng một **Tổng Riemann** (công thức thứ hai).

Tức là, chúng em chia tia sáng thành N_s đoạn nhỏ và lấy mẫu tại các điểm đó."

(Chỉ tay vào phần Mathematical Insight)

2. Tại sao ít mẫu lại gây lỗi? (The Insight):

"Tại đây, nhóm muốn nhấn mạnh một phân tích toán học quan trọng giải thích cho thất bại trong phần thực nghiệm sắp tới.

Trong công thức Riemann, độ mờ (opacity) của vật thể được tính bởi cụm $(1 - e^{-\sigma\delta_i})$.

Trong đó δ_i là khoảng cách giữa các điểm mẫu.

Khi chúng em giảm số mẫu N_s xuống quá thấp (ví dụ bằng 8):

- Khoảng cách δ_i giữa các điểm mẫu trở nên quá lớn.
- Sai số tích lũy tăng vọt.
- Hệ quả là tổng độ mờ không bao giờ đạt được giá trị 1.0 (tức là vật thể rắn). Thay vào đó, nó tạo ra các vật thể bán trong suốt."

3. Kết luận:

"Đây chính là nguyên nhân toán học dẫn đến hiện tượng **'Cloudy Artifacts'** (vật thể mờ như đám mây) mà chúng ta sẽ thấy minh họa trực quan ở phần kết quả."

1. The Integral:

Công thức đầu tiên là:

$$C(r) = \int T(t)\sigma(r(t))c(r(t))dt$$

- $C(r)$: Màu sắc (color) của tia sáng đi qua không gian 3D theo một đường ray r. Đây là tổng hợp của tất cả các màu sắc được phát ra từ vật liệu trong không gian 3D mà tia sáng đi qua.
- $T(t)$: **Transmittance function** - Độ trong suốt của vật liệu tại điểm t dọc theo đường ray. Nó mô tả mức độ mà ánh sáng có thể xuyên qua một vật liệu ở vị trí t trên đường ray.
- $\sigma(r(t))$: **Volume density** - Mật độ thể tích tại điểm $r(t)$. Đây là độ mờ của vật liệu tại điểm đó, xác định mức độ ánh sáng bị hấp thụ hoặc tán xạ.
- $c(r(t))$: **Emitted color** - Màu sắc phát ra từ vật liệu tại điểm $r(t)$.
- dt: Một yếu tố vi phân dọc theo đường ray, dùng để tính toán tích phân trên tất cả các điểm mà tia sáng đi qua.

Mục đích: Công thức này biểu thị cách tính màu sắc tổng thể của một tia sáng khi nó đi qua không gian 3D. Ánh sáng bị tán xạ và hấp thụ ở mỗi điểm trong không gian, và tất cả các yếu tố này được tính tổng lại để tạo ra màu sắc cuối cùng.

2. Riemann Sum Approximation:

Công thức tiếp theo là:

$$\hat{C}(r) \approx \sum_{i=1}^{N_s} T_i \left(1 - e^{-\sigma_i \delta_i}\right) c_i$$

- $\hat{C}(r)$: Đây là **ước lượng màu sắc** của tia sáng tại điểm r sau khi sử dụng **phương pháp Riemann sum** (phương pháp số để ước lượng tích phân).
- T_i: Độ trong suốt (transmittance) tại điểm mẫu thứ i.
- σ_i: Mật độ thể tích tại điểm mẫu thứ i.
- δ_i: Khoảng cách giữa các điểm mẫu (tức là bước nhảy giữa các điểm mà tia sáng đi qua).
- c_i: Màu sắc phát ra tại điểm mẫu thứ i.

Mục đích: Phương trình này ước lượng màu sắc cuối cùng của tia sáng bằng cách tính tổng các màu sắc phát ra từ các điểm mẫu dọc theo đường ray. Thay vì tính tích phân chính xác, công thức này sử dụng phép **Riemann sum** để tính toán gần đúng màu sắc của tia sáng, giúp tiết kiệm tài nguyên tính toán.

Công thức R1 Gradient Penalty:

$$R_1(D) = \frac{\gamma}{2} \mathbb{E}_x \left[\|\nabla_x D(x)\|^2 \right]$$

Giải thích các thành phần:

- $R_1(D)$: Đây là **R1 gradient penalty** cho **discriminator (D)**. Đây là một hình phạt (penalty) được thêm vào hàm mất mát của **discriminator**, giúp điều chỉnh gradient của **D** trong quá trình huấn luyện.
- γ: Đây là hệ số **regularization** (hệ số điều chỉnh). Hệ số này quyết định mức độ ảnh hưởng của hình phạt vào quá trình huấn luyện. Nếu γ quá lớn, nó có thể làm mất đi sự ổn định trong quá trình huấn luyện, nếu quá nhỏ thì nó không có tác dụng đủ mạnh.
- $\mathbb{E}_x$: Là kỳ vọng (expectation) trên **phân phối dữ liệu thực** P_{data}. Điều này có nghĩa là hàm này được tính trên các mẫu dữ liệu thực x mà **discriminator** phải phân biệt.
- $\nabla_x D(x)$: Đây là **gradient của discriminator** $D(x)$ theo đầu vào x, tức là sự thay đổi của giá trị phân loại $D(x)$ khi có sự thay đổi trong đầu vào x.
- $\|\nabla_x D(x)\|^2$: Là **bình phương độ lớn của gradient**, thể hiện mức độ thay đổi của **discriminator** khi đầu vào thay đổi. Nếu gradient lớn, điều đó có nghĩa là **discriminator** đang thay đổi rất mạnh khi dữ liệu thay đổi, điều này có thể dẫn đến sự không ổn định trong quá trình huấn luyện.