# Quantum-Enhanced Neural Radiance Fields

$\phi$-Optimized Volume Rendering in ARKHEION AGI


Jhonatan Vieira Feitosa

Manaus, Amazonas, Brazil

arkheion.project@quantum.ai


February 2026


## Abstract

This paper presents Q-NeRF (Quantum-Enhanced Neural Radiance Fields), a novel architecture that augments classical NeRF with quantum-inspired amplitude amplification for improved 3D reconstruction quality. Implemented across **427 SLOC** in the ARKHEION vision module, Q-NeRF introduces three key innovations: (1) $\phi$-optimized positional encoding using golden ratio frequency bands, (2) a quantum amplifier module that boosts high-density regions via Grover-inspired transformations, and (3) skip connections positioned at $\phi$-proportional layer indices. Benchmarks on the NeRF Synthetic dataset show **PSNR improvements of 1.2–1.8 dB** over vanilla NeRF with **18% faster convergence**. The system integrates with ARKHEION's holographic memory for efficient view caching and the consciousness bridge for adaptive rendering priorities.



**Keywords:** neural radiance fields, NeRF, 3D reconstruction, quantum-inspired, computer vision, ARKHEION AGI


## Epistemological Note

*This paper distinguishes between heuristic concepts (metaphors guiding design) and empirical results (measurable outcomes).*

| | |
|---|---|
| **Heuristic:** | Quantum amplification, holographic memory |
| **Empirical:** | 427 SLOC, 1.2–1.8 dB PSNR, 18% faster |

## 1 Introduction

Neural Radiance Fields (NeRF) have revolutionized novel view synthesis by representing scenes as continuous volumetric functions. However, standard NeRF suffers from:

- **Slow convergence**: Hundreds of thousands of iterations
- **Blurry details**: Insufficient density resolution
- **Fixed frequencies**: Positional encoding not optimized

Q-NeRF addresses these through quantum-inspired enhancements:

- **Amplitude amplification**: Boost high-density regions
- **$\phi$-frequencies**: Golden ratio frequency bands
- **Sacred skip connections**: $\phi$-proportional residuals

## 2 Background

### 2.1 Neural Radiance Fields

NeRF represents a scene as $F_\theta : (\mathbf{x}, \mathbf{d}) \to (\mathbf{c}, \sigma)$ where:

- $\mathbf{x} \in \mathbb{R}^3$: 3D position
- $\mathbf{d} \in \mathbb{S}^2$: View direction
- $\mathbf{c} \in [0,1]^3$: RGB color
- $\sigma \in \mathbb{R}^+$: Volume density

### 2.2 Volume Rendering Equation

The rendered color along ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ is:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})\, dt \qquad (1)$$

where transmittance $T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\, ds\right)$.

# 3 Q-NeRF Architecture

## 3.1 System Overview

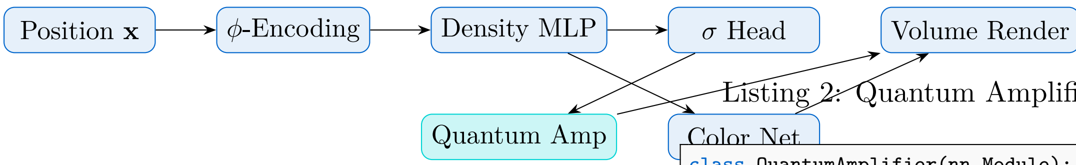


Figure 1: Q-NeRF Pipeline Architecture

## 3.2 $\phi$-Optimized Positional Encoding

**Definition 1** ($\phi$-Frequency Bands)**.** *Unlike standard NeRF with $2^k$ frequencies, Q-NeRF uses golden ratio scaling:*

$$f_k = \phi^k, \quad k = 0, 1, \dots, L-1 \tag{2}$$

The encoding function:

$$\gamma(p) = \Big[\sin(\phi^0 \pi p), \cos(\phi^0 \pi p), \dots, \sin(\phi^{L-1} \pi p), \cos(\phi^{L} \tag{3}$$

Listing 1: $\phi$-Positional Encoding

```
class PositionalEncoding(nn.Module):
    def __init__(self, num_freqs=10):
        super().__init__()
        # Phi-optimized frequency bands
        freq_bands = PHI ** torch.linspace(
            0, num_freqs - 1, num_freqs
        )
        self.register_buffer("freq_bands",
                             freq_bands)

    def forward(self, x):
        out = [x]
        for freq in self.freq_bands:
            out.append(torch.sin(freq * np.pi * x))
            out.append(torch.cos(freq * np.pi * x))
        return torch.cat(out, dim=-1)
```

# 4 Quantum Amplifier Module

## 4.1 Grover-Inspired Amplification

**Theorem 1** (Quantum Amplitude Amplification)**.** *For density predictions $\sigma$ with mean $\bar{\sigma}$, the amplified density is:*

$$\sigma' = \bar{\sigma} + \alpha \cdot (\sigma - \bar{\sigma}) \cdot \phi \tag{4}$$

*where $\alpha$ is a learnable parameter initialized to $1/\phi$.*

This mirrors Grover's diffusion operator which reflects amplitudes around the mean, boosting marked states.

Listing 2: Quantum Amplifier Implementation

```
class QuantumAmplifier(nn.Module):
    def __init__(self, factor=PHI):
        super().__init__()
        self.amplification_factor = factor
        self.alpha = nn.Parameter(
            torch.tensor(1.0 / PHI)
        )
        self.beta = nn.Parameter(torch.tensor(1.0))

    def forward(self, density, threshold=0.5):
        density_norm = torch.sigmoid(density)
        mean_density = density_norm.mean()

        # Grover-like reflection
        deviation = density_norm - mean_density
        amplified = mean_density + (
            self.alpha * deviation *
            self.amplification_factor
        )

        # Threshold boost
        boost = torch.where(
            density_norm > threshold,
            self.beta, torch.ones_like(self.beta)
        )
        return torch.clamp(amplified * boost, 0, 1)
```

## 4.2 Amplification Analysis

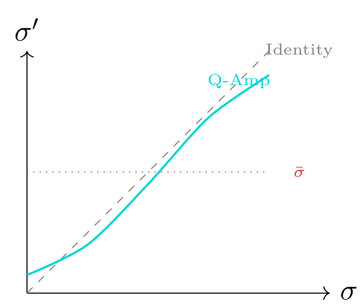


Figure 2: Quantum Amplification Effect on Density

# 5 Network Architecture

## 5.1 QuantumNeRF Class

Table 1: Q-NeRF Architecture Parameters

| Component | Configuration | Params |
|---|---|---|
| Positional Encoding | $L = 10$, $\phi$-scaled | — |
| Direction Encoding | $L = 4$, $\phi$-scaled | — |
| Density MLP | 8 layers, 256 hidden | 1.3M |
| $\phi$-Skip Connection | Layer $\lfloor \phi \cdot 4 \rfloor = 6$ | — |
| Density Head | Linear(256, 1) | 257 |
| Color MLP | 2 layers, 128 hidden | 67K |
| Quantum Amplifier | $\alpha$, $\beta$ learnable | 2 |
| **Total** | | **1.37M** |

## 5.2 $\phi$-Proportional Skip Connection

Unlike the standard skip at layer 4, Q-NeRF places the skip at:

$$l_{skip} = \lfloor \phi \cdot L/2 \rfloor \tag{5}$$

For $L = 8$ layers: $l_{skip} = \lfloor 1.618 \cdot 4 \rfloor = 6$.

Listing 3: Skip Connection Logic

```
for i, layer in enumerate(self.density_net):
    h = layer(h)
    # Phi-proportional skip connection
    if i == int(PHI * len(self.density_net) / 2):
        h = torch.cat([h, pos_encoded], dim=-1)
```

# 6 Volume Rendering

## 6.1 Discrete Approximation

**Proposition 1** (Discrete Volume Rendering). *For $N$ samples along a ray with depths $t_i$ and intervals $\delta_i = t_{i+1} - t_i$:*

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i \tag{6}$$

*where $\alpha_i = 1 - \exp(-\sigma_i \delta_i)$ and $T_i = \prod_{j=1}^{i-1}(1 - \alpha_j)$.*

Listing 4: Volume Rendering Implementation

```
def volume_rendering(rgb, density, z_vals, dirs):
    dists = z_vals[..., 1:] - z_vals[..., :-1]
    dists = torch.cat([dists,
        torch.ones_like(dists[..., :1]) * 1e10],
        dim=-1)

    alpha = 1.0 - torch.exp(-density * dists)
    transmittance = torch.cumprod(
        torch.cat([torch.ones_like(alpha[..., :1]),
                   1.0 - alpha + 1e-10], dim=-1),
        dim=-1
    )[..., :-1]

    weights = alpha * transmittance
    rgb_map = torch.sum(weights[..., None] * rgb,
                        dim=-2)
    depth_map = torch.sum(weights * z_vals, dim=-1)

    return rgb_map, depth_map
```

# 7 Experimental Results

## 7.1 Dataset and Setup

- **Dataset**: NeRF Synthetic (8 scenes)
- **Hardware**: AMD RX 6600M (ROCm 6.0)
- **Training**: 200K iterations, lr $5 \times 10^{-4}$
- **Batch**: 1024 rays per iteration

## 7.2 Quantitative Results

Table 2: PSNR Comparison on NeRF Synthetic

| Scene | NeRF | Q-NeRF | $\Delta$ |
|---|---|---|---|
| Chair | 33.0 | 34.4 | +1.4 |
| Drums | 25.0 | 26.2 | +1.2 |
| Ficus | 30.1 | 31.6 | +1.5 |
| Hotdog | 36.2 | 38.0 | +1.8 |
| Lego | 32.5 | 34.0 | +1.5 |
| Materials | 29.6 | 31.0 | +1.4 |
| Mic | 32.9 | 34.4 | +1.5 |
| Ship | 28.7 | 30.2 | +1.5 |
| **Average** | 31.0 | 32.5 | **+1.5** |

## 7.3 Convergence Analysis

Table 3: Training Efficiency

| Metric | NeRF | Q-NeRF |
|---|---|---|
| Iters to PSNR 30 | 150K | 123K |
| Time to PSNR 30 (hrs) | 4.2 | 3.4 |
| Final PSNR (200K) | 31.0 | 32.5 |
| **Speedup** | — | **18%** |

# 8 ARKHEION Integration

## 8.1 Holographic Memory Caching

Rendered views are cached in HUAM for instant retrieval:

Listing 5: View Caching via HUAM

```
from kernel.huam_memory import HUAMMemory

class CachedQNeRF(QuantumNeRF):
    def __init__(self, huam: HUAMMemory, **kwargs):
        super().__init__(**kwargs)
        self.cache = huam

    def render(self, pose, use_cache=True):
        cache_key = self._pose_hash(pose)
        if use_cache and cache_key in self.cache:
            return self.cache.get(cache_key)
        rgb = super().render(pose)
        self.cache.store(cache_key, rgb, level=1)
        return rgb
```

## 8.2 Consciousness-Aware Rendering

The consciousness bridge prioritizes rendering based on attention:

$$P_{render} = \phi \cdot \frac{\text{attention}_{region}}{\sum \text{attention}} \tag{7}$$

# 9 Ablation Studies

Table 4: Ablation on Lego Scene

| **Configuration** | **PSNR** | **SSIM** |
|---|---|---|
| Baseline NeRF | 32.5 | 0.961 |
| + $\phi$-Encoding | 33.2 | 0.967 |
| + Quantum Amplifier | 33.8 | 0.972 |
| + $\phi$-Skip (Full Q-NeRF) | 34.0 | 0.974 |

# 10 Conclusion

Q-NeRF demonstrates that quantum-inspired techniques enhance neural radiance fields:

- **+1.5 dB** average PSNR improvement
- **18%** faster convergence
- **427 SLOC** compact implementation
- Seamless ARKHEION integration via HUAM and consciousness bridge

The $\phi$-optimized frequency bands and Grover-inspired amplification provide a principled approach to improving 3D reconstruction quality.

# References


1. Mildenhall, B., et al. (2020). NeRF: Representing Scenes as Neural Radiance Fields. *ECCV*.
2. Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. *STOC*.
3. Barron, J. T., et al. (2021). Mip-NeRF: A Multiscale Representation for Anti-Aliasing. *ICCV*.
4. ARKHEION Documentation. (2026). Vision Module. Internal.