BACON: Band-limited Coordinate Networks for Multiscale Scene Representation

Supplemental Material

David B. Lindell Dave Van Veen Jeong Joon Park Gordon Wetzstein
Stanford University
http://computationalimaging.org/publications/bacon

Contents

1. Supplemental Derivations

1.1. Deriving the Number of Sines

Lemma 1.1.1. *The product of two sines is the sum of two sines with frequencies corresponding to the sum and difference of the initial frequencies.*

$$\sin(\omega_1 x + \phi_1) \cdot \sin(\omega_2 x + \phi_2) = \frac{1}{2}\Big[\sin\big((\omega_1 + \omega_2)\cdot x + \phi_1 + \phi_2 - \pi/2\big) + \sin\big((\omega_1 - \omega_2)\cdot x + \phi_1 - \phi_2 + \pi/2\big)\Big] \quad (1)$$

Proof. Without loss of generality, let $\sin(a) = \sin(\omega_1 x + \phi_1)$ and $\sin(b) = \sin(\omega_2 x + \phi_2)$. Then,

$$\begin{aligned}
\sin(a)\cdot\sin(b) &= \frac{e^{ja} - e^{-ja}}{2j}\cdot\frac{e^{jb} - e^{-jb}}{2j} && (2)\\
&= \frac{e^{j(a+b)} + e^{-j(a+b)} - e^{j(a-b)} - e^{-j(a-b)}}{-4} && (3)\\
&= -\frac{1}{2}\frac{e^{-j\pi/2}}{-j}\frac{e^{j(a+b)} + e^{-j(a+b)}}{2} + \frac{1}{2}\frac{e^{j\pi/2}}{j}\frac{e^{j(a-b)} + e^{-j(a-b)}}{2} && (4)\\
&= \frac{1}{2}\frac{e^{j(a+b-\pi/2)} + e^{-j(a+b-\pi/2)-j\pi}}{2j} + \frac{1}{2}\frac{e^{j(a-b+\pi/2)} + e^{-j(a-b+\pi/2)+j\pi}}{2j} && (5)\\
&= \frac{1}{2}\frac{e^{j(a+b-\pi/2)} - e^{-j(a+b-\pi/2)}}{2j} + \frac{1}{2}\frac{e^{j(a-b+\pi/2)} - e^{-j(a-b+\pi/2)}}{2j} && (6)\\
&= \frac{1}{2}\Big[\sin(a+b-\pi/2) + \sin(a-b+\pi/2)\Big] && (7)\\
& && (8)
\end{aligned}$$

□

Theorem 1.1.1. *The number of sines represented by a multiplicative filter network is given as*

$$N_{\text{sine}}^{(N_{\text{L}})} = \sum_{i=0}^{N_{\text{L}}-1} 2^i d_{\text{h}}^{i+1},$$

where N_{L} is the number of layers and d_{h} is the number of hidden features in the network.

Proof. Consider a $N_{\text{L}} = 1$ layer network, given as

$$\mathbf{z}_0 = \sin(\boldsymbol{\omega}_0\mathbf{x} + \boldsymbol{\phi}_0) \quad (9)$$

This expression is a vector of sines, and the proof follows trivially for $N_{\text{L}} = 1$; the number of sines represented by this network is exactly d_{h} because we have $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ and $\boldsymbol{\omega}_0 \in \mathbb{R}^{d_{\text{h}} \times d_{\text{in}}}$.

To build intuition, we also analyze the case of $N_{\text{L}} = 2$, which gives

$$\begin{aligned}
\mathbf{z}_1 &= \sin(\boldsymbol{\omega}_1\mathbf{x} + \boldsymbol{\phi}_1) \circ [\mathbf{W}_1 \sin(\boldsymbol{\omega}_0\mathbf{x} + \boldsymbol{\phi}_0) + \mathbf{b}_1] && (10)\\
&= \sin(\boldsymbol{\omega}_1\mathbf{x} + \boldsymbol{\phi}_1) \circ [\mathbf{W}_1 \sin(\boldsymbol{\omega}_0\mathbf{x} + \boldsymbol{\phi}_0)] + \sin(\boldsymbol{\omega}_1\mathbf{x} + \boldsymbol{\phi}_1) \circ \mathbf{b}_1 && (11)\\
& && (12)
\end{aligned}$$

where

$$\begin{aligned}
&\sin(\boldsymbol{\omega}_1\mathbf{x} + \boldsymbol{\phi}_1) \circ [\mathbf{W}_1 \sin(\boldsymbol{\omega}_0\mathbf{x} + \boldsymbol{\phi}_0)] && (13)\\
&= \begin{bmatrix} \sin(\boldsymbol{\omega}_1^{(1)}\mathbf{x} + \phi_1^{(1)}) \\ \vdots \\ \sin(\boldsymbol{\omega}_1^{(d_{\text{h}})}\mathbf{x} + \phi_1^{(d_{\text{h}})}) \end{bmatrix} \circ \begin{bmatrix} W_1^{(1,1)} \sin(\boldsymbol{\omega}_0^{(1)}\mathbf{x} + \phi_0^{(1)}) + \cdots + W_1^{(1,d_{\text{h}})} \sin(\boldsymbol{\omega}_0^{(d_{\text{h}})}\mathbf{x} + \phi_0^{(d_{\text{h}})}) \\ \vdots \\ W_1^{(d_{\text{h}},1)} \sin(\boldsymbol{\omega}_0^{(1)}\mathbf{x} + \phi_0^{(1)}) + \cdots + W_1^{(d_{\text{h}},d_{\text{h}})} \sin(\boldsymbol{\omega}_0^{(d_{\text{h}})}\mathbf{x} + \phi_0^{(d_{\text{h}})}). \end{bmatrix} && (14)
\end{aligned}$$

Given that the Hadamard product between each row in the above expression results in a doubling of the number of sines (**Lemma 1.1.1**), the entire Hadamard product results in a vector containing $2d_{\text{h}}^2$ sines. The remaining term $\sin(\boldsymbol{\omega}_1\mathbf{x}+\boldsymbol{\phi}_1)\circ\mathbf{b}_1$ contributes an additional d_{h} sines, for a total $2d_{\text{h}}^2 + d_{\text{h}}$ sines in the $N_{\text{L}} = 2$ layer network. In general, the linear component $\mathbf{W}_i$ of each layer multiplies the total number of sines by d_{h} and the Hadamard product with the sine doubles this by a factor of 2. An additional d_{h} sines are contributed by the bias term.

Now, assume we have a network with $N_{\text{L}} = k$ layers. Let $L_i(\mathbf{z}) = \mathbf{W}_i\mathbf{z} + \mathbf{b}_i$ and let $g_i(\mathbf{x}) = \sin(\boldsymbol{\omega}_i\mathbf{x} + \boldsymbol{\phi}_i)$. Then we have

$$\mathbf{z}_{k-1} = g_{k-1}(\mathbf{x}) \circ (L_{k-1}(g_{k-2}(\mathbf{x}) \circ (\ldots (g_2(\mathbf{x}) \circ (\overbrace{L_2(g_1(\mathbf{x}) \circ (L_1 \underbrace{g_0(\mathbf{x})}_{d_{\text{h}}}))}^{d_{\text{h}}(2d_{\text{h}}^2+d_{\text{h}})}))\ldots)))), \tag{15}$$

$d_{\text{h}}+2d_{\text{h}}(2d_{\text{h}}^2+d_{\text{h}})$ (over the whole expression); d_{h}^2 (under L_1); $2d_{\text{h}}^2+d_{\text{h}}$ (under L_2)

where the brackets indicate the number of sines for successive terms, revealing the following recursion for the number of sines in each layer.

$$N_{\text{sine}}^{(1)} = d_{\text{h}} \tag{16}$$

$$N_{\text{sine}}^{(i+1)} = d_{\text{h}} + 2d_{\text{h}}N_{\text{sine}}^{(i)} \tag{17}$$

To complete a proof by induction let us assume that the theorem holds for $N_{\text{L}} = k$, and we have

$$N_{\text{sine}}^{(k+1)} = d_{\text{h}} + 2d_{\text{h}}N_{\text{sine}}^{(k)} \tag{18}$$

$$= d_{\text{h}} + 2d_{\text{h}} \sum_{i=0}^{k-1} 2^i d_{\text{h}}^{i+1} \tag{19}$$

$$= d_{\text{h}} + \sum_{i=0}^{k-1} 2^{i+1} d_{\text{h}}^{i+2} \quad \text{let } j = i + 1. \tag{20}$$

$$= d_{\text{h}} + \sum_{j=1}^{k} 2^j d_{\text{h}}^{j+1} \tag{21}$$

$$= \sum_{j=0}^{k} 2^j d_{\text{h}}^{j+1} \tag{22}$$

which is the original result, completing the proof. □

Corollary 1.1.1. *If we remove the bias layers from each L_i, then we remove the additional of d_h from the recursion and it becomes*

$$\tilde{N}_{\text{sine}}^{(1)} = d_h \tag{23}$$

$$\tilde{N}_{\text{sine}}^{(i+1)} = 2d_h N_{\text{sine}}^{(i)}, \tag{24}$$

such that

$$\tilde{N}_{\text{sine}}^{(N_{\text{L}})} = 2^{N_{\text{L}}-1} d_h^{N_{\text{L}}}. \tag{25}$$

1.2. Deriving the Distribution of Frequencies

Lemma 1.2.1. *Let X_i, $i = 1, 2, \ldots$ be a sequence of independent, identically distributed random variables. Then, let N be a discrete random variable that is independent of X_i and takes on values $N > 0$. Now, define the compound random variable*

$$S_N = \sum_{i=0}^{N-1} X_i, \tag{26}$$

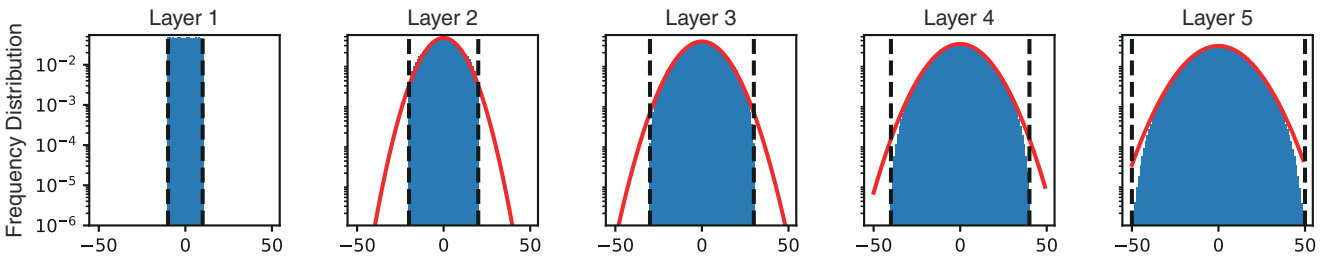


Figure 1. Empirical evaluation of distribution of frequencies. We plot the distribution of frequencies at the output of each layer in a 5-layer network with hidden features $d_{\text{h}} = 1024$ bandwidth B_i equal to 10 (averaged over 1000 network realizations). The output of the first layer is uniformly distributed and successive layers increasingly approximate the normal distribution according to the Central Limit Theorem. A normal distribution with variance calculated using **Theorem 1.2.1** is plotted in red and closely matches the observed distribution.

The variance of S_N is given by

$$\text{Var}(S_N) = E[N]\,\text{Var}(X_1) + \text{Var}(N)E[X_1]^2 \tag{27}$$

Proof. The proof follows from the law of total variance (see, e.g., p. 286 of Chatfield and Theobald [5]). □

Lemma 1.2.2. *Central Limit Theorem. Let $X_1, \ldots, X_n$ be a sequence of independent and identically distributed random variables with finite mean and variance, μ and σ^2, respectively, and let $S_n = \sum_{i=1}^{n} X_i$. For large n, S_n approximates the normal distribution with $E[s_n] = n\mu$ and $\text{Var}(s_n) = n\sigma^2$*

Proof. See Ash et al. [2]. □

Theorem 1.2.1. *The frequencies of* BACON *are approximately Gaussian distributed with variance equal to*

$$\left[\sum_{m=0}^{N_L-1} m \cdot \frac{2^{N_L-1-m} d_h^{N_L-m}}{\sum_{i=0}^{N_L-1} 2^i d_h^{i+1}}\right] \cdot \text{Var}(\boldsymbol{\omega}_i^{(j_i)}). \tag{28}$$

where $\text{Var}(\boldsymbol{\omega}_i^{(j_i)})$ is the variance of the initialized frequencies in each input sine layer g_i.

Proof. The overall idea is that, if the number of sines is in the network is given as $\sum_{i=0}^{N_{\text{L}}-1} 2^i d_{\text{h}}^{i+1}$ (**Theorem 1.1.1**), then it turns out that the ith element in this sum describes the number of sines whose frequency is the sum of $i+1$ random variables. Then, we can use **Lemma 1.2.1** and **Lemma 1.2.2** to derive the variance and distribution of the network frequencies.

First, let us show that the frequency of each sine represented by the network is itself a sum of random variables. We write an expression for the number of sines in the network $F_i(\boldsymbol{\omega})$ at frequency $\boldsymbol{\omega}$ directly after applying the Hadamard product with $g_i(\mathbf{x})$. Let $\delta(\boldsymbol{\omega})$ represent the Dirac delta function. Expanding on our previous results, we have that

$$F_0(\boldsymbol{\omega}) = \int_{-\infty}^{\infty} \underbrace{\sum_{j=0}^{d_{\text{h}}-1} \delta(\boldsymbol{\omega} - \boldsymbol{\omega}_0^{(j)})}_{f_0(\boldsymbol{\omega})} \,\mathrm{d}\boldsymbol{\omega}. \tag{29}$$

This expression simply places a delta function at the location of each frequency and the integral checks to see how many frequencies exist at the input parameter frequency, $\boldsymbol{\omega}$.

Now, recall the previous result that adding the next layer multiplies the number of frequencies by $2d_{\text{h}}$ and adds an additional d_{h} frequencies. We use the convolution operator $*$ to shift the frequencies of the previous layer according to

Lemma 1.1.1. The additional d_h frequencies result from the Hadamard product of the sine layer and the bias term from the previous layer. Thus we can give the following expression for the frequencies at layer $i+1$.

$$F_{i+1}(\boldsymbol{\omega}) = \int_{-\infty}^{\infty} f_i(\boldsymbol{\omega}) * \underbrace{\sum_{j=0}^{d_\text{h}-1} \left[\delta(\boldsymbol{\omega} - \boldsymbol{\omega}_{i+1}^{(j)}) + \delta(\boldsymbol{\omega} + \boldsymbol{\omega}_{i+1}^{(j)})\right]}_{\text{shifts spectrum according to } \textbf{Lemma 1.1.1}} + \underbrace{\sum_{j=0}^{d_\text{h}-1} \delta(\boldsymbol{\omega} - \boldsymbol{\omega}_{i+1}^{(j)})}_{\text{new frequencies from the bias term}} \quad \mathrm{d}\boldsymbol{\omega}. \tag{30}$$

If we ignore frequencies resulting from the bias terms in the above expression, we would have that there are $2^{N_\text{L}-1} d_\text{h}^{N_\text{L}}$ sines in an N_L layer network (**Theorem 1.1.1**) with output frequencies given as

$$\bar{\boldsymbol{\omega}} = \boldsymbol{\omega}_0^{(j_0)} + s_1 \boldsymbol{\omega}_1^{(j_1)} + \cdots + s_{N_\text{L}-1} \boldsymbol{\omega}_{N_\text{L}-1}^{(j_{N_\text{L}-1})} \tag{31}$$

for some $s_0, \ldots, s_{N_\text{L}-1} \in \{-1, 1\}$ and indices $j_0, \ldots, j_{N_\text{L}-1} \in \{0, 1, \ldots, d_\text{h} - 1\}$. Including the bias term at the ith layer (for $i > 0$) simply results in an additional $2^{N_\text{L}-1-i} d_\text{h}^{N_\text{L}-i}$ sines in the network, whose frequencies are a sum of $N_\text{L} - i$ terms (corresponding exactly to the terms of the sum in **Theorem 1.1.1**). Thus, frequencies represented by the network are drawn from a compound distribution (as in **Lemma 1.2.1**) since they can be the sum of from 1 to N_L random variables.

Now we can describe the distribution of the frequencies of the network. Let the variance of an element of $\boldsymbol{\omega}_i$ be given by $\text{Var}(\boldsymbol{\omega}_i^{(j_i)})$. Then, the output frequency $\bar{\boldsymbol{\omega}}$ is a compound random variable such that

$$\bar{\boldsymbol{\omega}} = \boldsymbol{\omega}_M^{(j_M)} + \sum_{i=M+1}^{N_\text{L}-1} s_i \boldsymbol{\omega}_i^{(j_i)} \quad M \in \{0, \ldots, N_\text{L} - 1\} \tag{32}$$

and M is a random variable whose probability depends on the total number of frequencies in the network that contribute to each possible value of M. Specifically, using **Theorem 1.1.1** to evaluate the fraction of sines for each value of M gives

$$p_M(m) = \frac{2^{N_\text{L}-1-m} d_\text{h}^{N_\text{L}-m}}{\sum_{i=0}^{N_\text{L}-1} 2^i d_\text{h}^{i+1}}. \tag{33}$$

We can calculate the resulting variance using the law of total variance outlined in **Lemma 1.2.1**.

$$\begin{aligned}
\text{Var}(\bar{\boldsymbol{\omega}}) &= E[M]\,\text{Var}(\boldsymbol{\omega}_i^{(j_i)}) + \text{Var}(M) E[\boldsymbol{\omega}_i^{(j_i)}]^2 && && (34)\\
&= E[M]\,\text{Var}(\boldsymbol{\omega}_i^{(j_i)}) && \boldsymbol{\omega}_i^{(j_i)} \text{ zero mean} && (35)\\
&= \sum_{m=0}^{N_\text{L}-1} m \cdot p_M(m)\,\text{Var}(\boldsymbol{\omega}_i^{(j_i)}) && && (36)\\
&= \left[\sum_{m=0}^{N_\text{L}-1} m \cdot \frac{2^{N_\text{L}-1-m} d_\text{h}^{N_\text{L}-m}}{\sum_{i=0}^{N_\text{L}-1} 2^i d_\text{h}^{i+1}}\right] \cdot \text{Var}(\boldsymbol{\omega}_i^{(j_i)}) && && (37)
\end{aligned}$$

Finally, we note that $\bar{\boldsymbol{\omega}}$ becomes the sum of a large number of random variables as the number of hidden layers in the network increases, and so we can approximate the distribution as a Gaussian using the Central Limit Theorem (**Lemma 1.2.2**). We show that this holds empirically in Fig. 1, where we show the simulated distribution of frequencies in each layer of a 5-layer network with $d_\text{h} = 1024$ and $B_i = 10$, averaged over 1000 realizations. The distribution of frequencies is well-approximated by a Gaussian with the derived variance (especially for increasing layers). □

1.3. Initialization and Distribution of Activations

1.3.1 Preliminary Derivations

Lemma 1.3.1. *Let X and Y be two independent random variables with probability density functions f_X and f_Y. Then the probability density function $f_Z(z)$ of $Z = XY$ is given as*

$$f_Z(z) = \int_{-\infty}^{\infty} f_X(x) f_Y(z/x) \frac{1}{|x|} \mathrm{d}x. \tag{38}$$

Proof. See Grimmett and Stirzaker [8]. □

Theorem 1.3.1. *Let W, X, and P be independent random variables sampled from continuous uniform distributions as*

$$W \sim \mathcal{U}(-B, B) \tag{39}$$
$$X \sim \mathcal{U}(-0.5, 0.5) \tag{40}$$
$$P \sim \mathcal{U}(-\pi, \pi) \tag{41}$$

where $B \gg \pi$. Then let $Z = WX + P$. The probability density function $f_Z(z)$ of Z is approximately

$$f_Z(z) \approx \begin{cases} \frac{1}{B} \log\left(\frac{B}{|2z|}\right), & -B/2 \leq z \leq B/2 \\ 0, & \text{else} \end{cases}. \tag{42}$$

Proof. Let $\tilde{Z} = WX$. Then, $f_{\tilde{Z}}(z)$ is given as (**Lemma 1.3.1**):

$$f_{\tilde{Z}}(z) = \int_{-\infty}^{\infty} f_W(w) f_X(z/w) \frac{1}{|w|} \, \mathrm{d}w \tag{43}$$
$$= 2 \int_{0}^{\infty} f_W(w) f_X(z/w) \frac{1}{w} \, \mathrm{d}w \tag{44}$$
$$= \frac{1}{B} \int_{0}^{B} f_X(z/w) \frac{1}{w} \, \mathrm{d}w \tag{45}$$
$$f_X(z/w) = \begin{cases} 1 & -0.5 \leq z/w \leq 0.5 \\ 0 & \text{else} \end{cases} \tag{46}$$
$$= \begin{cases} 1 & w \leq -2z, \, w \geq 2z \\ 0 & \text{else} \end{cases} \tag{47}$$
$$= \frac{1}{B} \int_{\min(2z, B)}^{B} \frac{1}{w} \, \mathrm{d}w \tag{48}$$
$$= \frac{1}{B} \log(|w|) \Big|_{\min(2z, B)}^{B} \tag{49}$$
$$= \frac{1}{B} \log\left(\frac{B}{\min(|2z|, B)}\right) \tag{50}$$
$$= \begin{cases} \frac{1}{B} \log\left(\frac{B}{|2z|}\right), & -B/2 \leq z \leq B/2 \\ 0, & \text{else} \end{cases} \tag{51}$$

Now, $Z = WX + P = \tilde{Z} + P$ and $f_Z = f_{\tilde{Z}} * f_P$, where $*$ indicates convolution. For $B \gg \pi$, the support of f_P is sufficiently small that we can neglect the "broadening" effect of the convolution, such that $f_Z \approx f_{\tilde{Z}}$. □

Theorem 1.3.2. *With Z and B as defined in **Theorem 1.3.1**, we have that*

$$\text{Var}[\sin(Z)] \approx \frac{1}{2}\left[1 - \frac{\text{SI}(B)}{B}\right] \approx \frac{1}{2} \tag{52}$$

Proof.

$$\begin{aligned}
\text{Var}[\sin(Z)] &= \text{E}[\sin^2(Z)] && (53)\\
&= \frac{1}{2}(1 - \text{E}[\cos(2Z)]) && (54)\\
\text{E}[\cos(2Z)] &= \int_{-\infty}^{\infty} f_Z(z)\cos(2z)\,\mathrm{d}z && (55)\\
&\approx \frac{1}{B}\int_{-B/2}^{B/2} \log\left(\frac{B}{|2z|}\right)\cos(2z)\,\mathrm{d}z \qquad (\textbf{Theorem 1.3.1}) && (56)\\
&\quad \text{Integrate by parts: } \int f\,\mathrm{d}g = fg - \int \mathrm{d}f g && (57)\\
&\quad f = \log\left(\frac{B}{|2z|}\right),\ \mathrm{d}g = \cos(2z)\,\mathrm{d}z,\ \mathrm{d}f = -\frac{1}{z}\,\mathrm{d}z,\ g = \frac{1}{2}\sin(2z) && (58)\\
&= \frac{1}{B}\left[\frac{1}{2}\log\left(\frac{B}{|2z|}\right)\sin(2z) + \underbrace{\int \frac{\sin(2z)}{2z}\,\mathrm{d}z}_{\frac{1}{2}\text{SI}(2z)}\right]_{-B/2}^{B/2} && (59)\\
&= \frac{1}{2B}\left[\log\left(\frac{B}{|2z|}\right)\sin(2z) + \text{SI}(2z)\right]_{-B/2}^{B/2} && (60)\\
&= \frac{1}{2B}\left[\log\left(\frac{B}{|2z|}\right)\sin(2z) + \text{SI}(2z)\right]_{-B/2}^{B/2} && (61)\\
&= \frac{\text{SI}(B)}{B} \approx \frac{\pi}{2B},\ B \gg 0 && (62)\\
\Rightarrow \frac{1}{2}(1 - \text{E}[\cos(2Z)]) &= \frac{1}{2}\left(1 - \frac{\text{SI}(B)}{B}\right) && (63)\\
&\approx \frac{1}{2}\left(1 - \frac{\pi}{2B}\right),\ B \gg 0 && (64)\\
&\approx \frac{1}{2} && (65)
\end{aligned}$$

Where we used that $\text{SI}(x)$ is the sine integral function: $\text{SI}(x) = \int_0^x \frac{\sin(t)}{t}\,\mathrm{d}t$. □

Lemma 1.3.2. *The variance of the product of two random variables X and Y is given by*

$$\text{Var}[X \cdot Y] = \text{Var}[X] \cdot \text{Var}[Y] + \text{E}[Y]^2 \cdot \text{Var}[X] + \text{E}[X]^2 \cdot \text{Var}[Y] \tag{66}$$

Proof. Refer to Goodman [7]. □

1.3.2 Proof of the Initialization Scheme

Theorem 1.3.3. *Let the input to* BACON *be uniformly distributed in $[-0.5, 0.5]$ and the frequency $\boldsymbol{\omega}_i$ of each layer be uniformly distributed in $[-B_i, B_i]$ with $B_i \gg 0$. Then, let the linear layer weights $\mathbf{W}_i$ applied after the sine layers be distributed according to a random uniform distribution in the interval $[-\sqrt{6/d_h}, \sqrt{6/d_h}]$. The activations after each linear layer are standard normal distributed.*

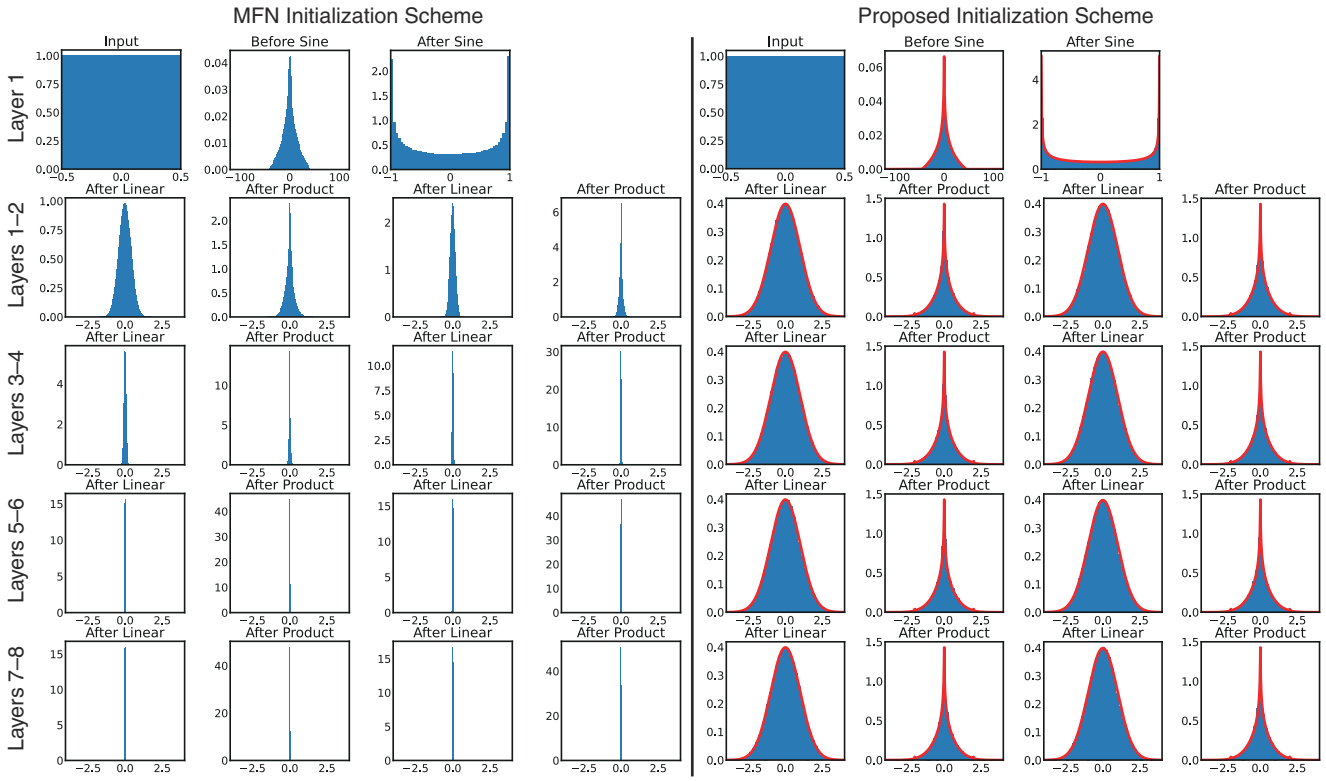


Figure 2. Empirical evaluation of initialization scheme. We show the default MFN initialization scheme (left) [6], and the proposed initialization scheme (right) for a network with 9 sine layers and 1024 hidden features (d_{h}). For our method we set the bandwidth B_i of each layer to an arbitrary value of 30π. We use the default settings of the MFN codebase, which initializes the linear layers $\mathbf{W}_i$ uniformly in $[-\sqrt{256/d_{\text{h}}}, \sqrt{256/d_{\text{h}}}]$ and sets the bandwidth to $B_i = 256\sqrt{d_{\text{in}}}$. Note that the proposed initialization scheme resolves the issue of the activation magnitude becoming extremely small with increased network depth. We also show that the distribution of activations closely matches our derivations, with the analytical expressions plotted in red.

Proof. A sketch of the proof is as follows.

- The variance of the output of each sine layer $g_i(\mathbf{x})$ is approximately 0.5 (**Theorem 1.3.2**).
- The output after applying the first linear layer $\mathbf{W}_1 g_0(\mathbf{x})$ is standard normal distributed (neglecting the effect of the bias). We have that the ith output is $\sum_{j=0}^{d_{\text{h}}-1} \mathbf{W}_1^{(i,j)} g_0(\mathbf{x})^{(j)}$, with $\text{Var}\left(\mathbf{W}_1^{(i,j)} g_0(\mathbf{x})^{(j)}\right) = \text{Var}\left(\mathbf{W}_1^{(i,j)}\right) \cdot \text{Var}\left(g_0(\mathbf{x})^{(j)}\right) = \frac{1}{12}\left(2\sqrt{6/d_{\text{h}}}\right)^2 \cdot \frac{1}{2} = 1/d_{\text{h}}$ (**Lemma 1.3.2**). Then the entire sum has variance $d_{\text{h}} \cdot 1/d_{\text{h}} = 1$, and is normal distributed by the Central Limit Theorem (**Lemma 1.2.2**).
- The output of the Hadamard product $g_1(\mathbf{x}) \circ (\mathbf{W}_1 g_0(\mathbf{x}) + \mathbf{b}_1)$ has variance $\text{Var}(g_1(\mathbf{x})) \cdot \text{Var}(\mathbf{W}_1 g_0(\mathbf{x}) + \mathbf{b}_1) \approx \frac{1}{2} \cdot 1 = \frac{1}{2}$ (**Lemma 1.3.2**).
- The distribution after applying the next linear layer $\mathbf{W}_{i+1}$ is again standard normal (using the same steps of taking the product of the variances and applying the Central Limit Theorem), and the same steps as above can be repeated to show the distributions are standard normal after each linear layer.

□

1.3.3 Empirical Evaluation

We empirically evaluate the initialization scheme and derivations by showing plots of the distributions of activations for BACON with 9 sine layers, 1024 hidden features (d_{h}), and an arbitrary bandwidth $B_i = 30\pi$. The plot is shown in Fig. 2, and

we overlay the analytical expression for the distribution at each intermediate output of the network. We also compare to the conventional MFN initialization proposed by Fathony et al. [6] using the publicly available implementation[1]. Their proposed method initializes the linear layers $\mathbf{W}_i$ to a random uniform value in $[-\sqrt{256/d_{\text{h}}}, \sqrt{256/d_{\text{h}}}]$ and sets the bandwidth to $B_i = 256\sqrt{d_{\text{in}}}$. We find that this causes the magnitude of the activations to decrease significantly with increasing network size, leading to vanishing gradients. There are five different distributions that can be observed at intermediate outputs of the network with our proposed initialization scheme:

1. The input to the network is uniformly distributed in [-0.5, 0.5].

2. The output of the linear layers applied directly to the input (that is, $\boldsymbol{\omega}_i\mathbf{x}+\boldsymbol{\phi}_i$) is distributed as derived in **Theorem 1.3.1**.

3. We find empirically that the output of $g_i(\mathbf{x}) = \sin(\boldsymbol{\omega}_i\mathbf{x} + \boldsymbol{\phi}_i)$ is approximately arcsine distributed and the variance is approximately $1/2$ as derived in **Theorem 1.3.2**. In Fig. 2, we show the empirical distribution together with a plot of the arcsine distribution with support over $[-1, 1]$ given by the probability density function $f_X(x) = \frac{1}{\pi\sqrt{1-x^2}}$. Note the close correspondence of the arcsine distribution and the observed histogram of activations in Fig. 2. We believe the connection to the arcsine distribution is related to previous work, which shows that the sine of uniform and normal random variables are both approximately arcsine distributed [12].

4. The output of $\mathbf{W}_1 g_0(\mathbf{x}) + \mathbf{b}_1$ is standard normal distribution as described in **Theorem 1.3.3**. Also, the outputs of other linear layers $\mathbf{W}_i$ with bias $\mathbf{b}_i$ are standard normal distributed.

5. The output of the Hadamard product $g_i(\mathbf{x}) \circ (\mathbf{W}_i\mathbf{z}_{i-1} + \mathbf{b}_i)$ is the product of an (approximately) arcsine distribution and a standard normal distribution. The distribution of a product of random variables is described by **Lemma 1.3.1**, and the variance of this distribution is approximately equal to 1/2 (**Lemma 1.3.2**). We numerically calculate the probability density function for the product of a standard normal and arcsine distribution according to **Lemma 1.3.1** and find that this approximates the empirical distribution as shown in Fig. 2.

[1] https://github.com/boschresearch/multiplicative-filter-networks

2. Supplemental Results

2.1. Images

We evaluate BACON on an image fitting task and compare its performance to three other methods: a ReLU network using Gaussian Fourier Features positional encoding (PE) [14] and SIREN [12], both supervised at 256×256 (1×) resolution, and a ReLU network using integrated PE (adapted from Mip-NeRF) [3] with supervision at 1/4, 1/2, and 1× resolutions. BACON is supervised at a single scale (1×) and learns a multiscale decomposition. All networks contain 4 hidden layers with 256 hidden features and are trained as described in the main text.

We perform a quantitative evaluation of the image fitting performance by training on a dataset of 16 randomly selected images from the DIV2K dataset and reporting the peak signal to noise ratio (PSNR) and structural similarity index measure (SSIM). We resize center crops of the images to 256×256 resolution and then fit a model to the grid of pixels for each image. We evaluate the performance on this training set of pixels as well as an offset validation grid of 256×256 pixels whose values are bilinearly interpolated. Quantitative results in Table 1 demonstrate that all methods fit the training set to well over 30 dB PSNR at at 256×256 (1×) resolution. All methods perform similarly on the validation set. Remarkably, BACON demonstrates similar performance to the single-scale representations while simultaneously representing all output scales.

Additional quantitative image results are shown in Table 2, evaluated across multiple scales. Here, we train the network in the same fashion as above, but evaluate on a 1/4 (64 × 64) or 1/2 (128 × 128) resolution coordinate grid or a 4× upsampled grid (1024 × 1024). We compare the network outputs to a bilinearly downsampled image or a high-resolution ground truth image in the case of 4× upsampling. BACON and integrated positional encoding show the best performance for the low-resolution images while all methods perform similarly for upsampling.

Fig. 3 shows all output resolutions (1/4, 1/2, 1, and 4×) for the result shown in the main text. Fig. 4 shows additional results on center-cropped images from the DIV2K dataset [1]. In all results, Fourier Features and SIREN fit to a single scale and show aliasing when subsampled, i.e. at 1/4 and 1/2 resolution. Integrated PE learns reasonable anti-aliasing as it is explicitly supervised on anti-aliased pixel values. The band-limited nature of BACON allows it to closely represent a low-pass filtered image while only explicitly supervising at 1× resolution. At 4× resolution, all methods except BACON show high-frequency artifacts.

Fig. 5 shows an experiment where we compare BACON and the network with a normal and low-pass filtered version of integrated position encoding at the 4× upsampled resolution. The integrated positional encoding result contains spurious high-frequency details and artifacts from aliasing since the bandwidth of the network is not constrained. Since aliasing corrupts the low-frequency components, these artifacts cannot be removed by applying a low-pass filter.

Deep BACON. We compare deep 8- and 16-layer versions of BACON with the proposed initialization scheme and MFNs with the original initialization scheme. For the image fitting task, we find that an 8-layer BACON fits the lighthouse image shown in the main paper to 38.8 dB PSNR versus 29.8 dB PSNR for an 8-layer MFN. For 16 layers, BACON fits the image to 37.4 dB while the MFN architecture fails to optimize due to numerical instabilities. We show convergence plots of the PSNR in Fig. 6.

Scale Interpolation. Interpolating between the discrete output scales allows a kind of continuous output scale to be achieved, similar to the trilinear filtering used to render from mimaps [15]. To illustrate this effect, we sample BACON at all output scales on the same 256×256 resolution grid, and then linearly interpolate between resulting images. Results are shown in Fig. 7. Note that since linear interpolation is used, there is a discontinuous appearance of high-frequency Fourier coefficients when moving from one scale to the next. Still, this simple technique allows blending between scales.

	# Params.	Training PSNR	Training SSIM	Validation PSNR	Validation SSIM
Fourier Features	264K	37.362 ± 2.544	0.976 ± 0.009	**29.771** ± 1.410	**0.931** ± 0.022
SIREN	265K	**41.851** ± 2.084	**0.987** ± 0.006	28.927 ± 1.756	0.922 ± 0.034
Integrated PE	274K	33.092 ± 2.219	0.930 ± 0.027	29.505 ± 1.498	0.901 ± 0.025
BACON	268K	38.871 ± 1.727	0.979 ± 0.005	29.266 ± 1.632	0.922 ± 0.023

Table 1. Quantitative evaluation (mean ± standard deviation) for image fitting. For BACON and Integrated Positional Encoding we compare to the highest resolution output.

Scale	Method	# Params.	PSNR	SSIM
1/4	Fourier Features	264K	24.484 ± 1.636	0.858 ± 0.055
	SIREN	265K	24.063 ± 1.832	0.843 ± 0.069
	Integrated PE	274K	**36.819** ± 1.697	**0.984** ± 0.007
	BACON	67K	31.179 ± 1.890	0.948 ± 0.011
1/2	Fourier Features	264K	30.830 ± 1.462	0.955 ± 0.017
	SIREN	265K	29.474 ± 2.024	0.942 ± 0.033
	Integrated PE	274K	33.020 ± 1.596	**0.959** ± 0.013
	BACON	199K	**33.140** ± 1.711	**0.959** ± 0.009
4×	Fourier Features	264K	25.909 ± 3.161	0.722 ± 0.122
	SIREN	265K	**26.198** ± 3.255	**0.740** ± 0.115
	Integrated PE	274K	24.530 ± 2.477	0.667 ± 0.127
	BACON	268K	25.967 ± 2.869	0.731 ± 0.108

Table 2. Quantitative evaluation (mean ± standard deviation) for image fitting, evaluated at multiple scales.

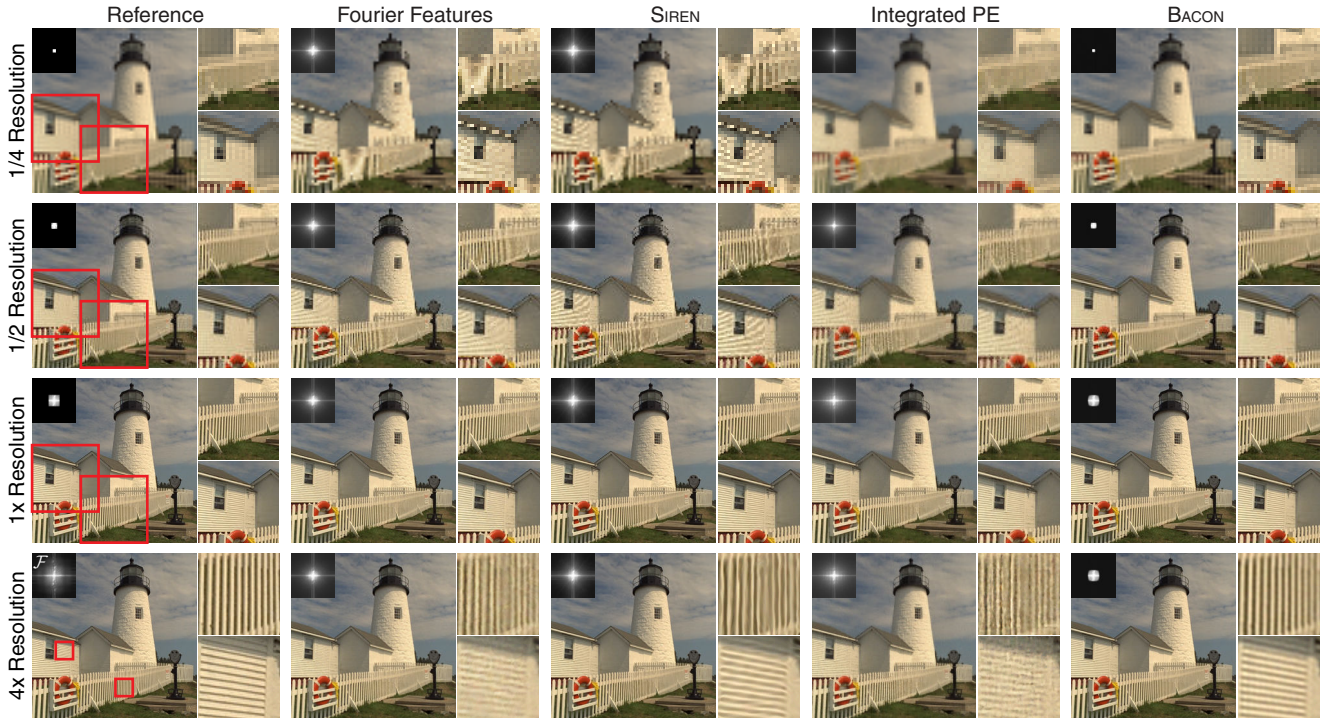


Figure 3. Image fitting results. We train a ReLU network using Fourier features (FF) positional encoding (PE) [14] and a SIREN [12] to fit an image at 256×256 (1×) resolution, and evaluate the models at 64×64 (1/4), 128×128 (1/2), 256×256 (1×), and 1024×1024 (4×) resolution. Since these methods fit to a single scale, we see aliasing at lower resolutions, and high-frequency artifacts at 4× resolution (see insets). We train a ReLU network with integrated PE [3] with supervision at 1/4, 1/2, and 1× resolutions. While this network learns anti-aliasing at low resolutions, inference at the unsupervised 4× resolution yields artifacts. Finally, BACON is supervised at a single scale, learning band-limited outputs that closely match low-pass filtered reference images (see left column, and Fourier spectra insets). All methods achieve an accurate fit at 1× resolution with PSNRs of 37.838 dB (FF PE), 41.513 dB (SIREN), 34.105 dB (Integrated PE), and 40.314 dB (BACON).

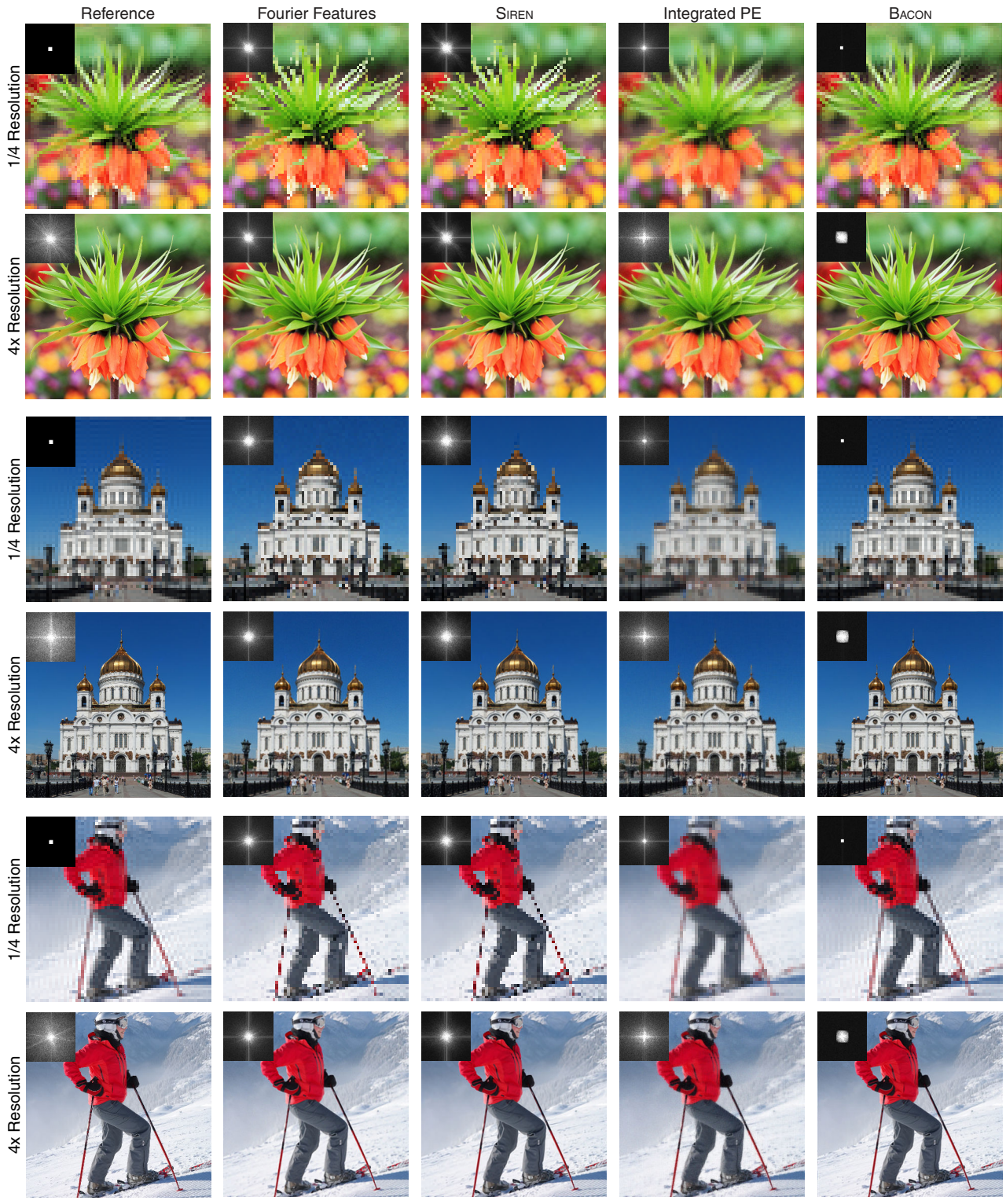


Figure 4. Supplemental image fitting results. We show the output of baseline methods and BACON fit to a subset of images from the DIV2K dataset. Similar to previous results, we observe aliasing in baseline methods when subsampling to lower resolutions, and artifacts in 4× supersampled outputs. BACON produces anti-aliased outputs at low-resolution and interpretable upsampled results via band-limited interpolation.

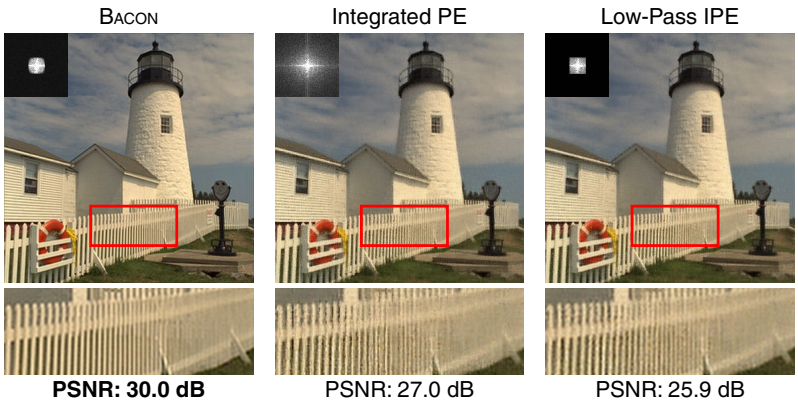


Figure 5. Applying low-pass filter to output of integrated positional encoding (IPE) network. Since IPE networks are not band limited, there are artifacts in the output when upsampling at 4× resolution (middle column). BACON (left column) is band limited and does not exhibit these artifacts. Spurious high frequency oscillations in the IPE network are aliased onto low-frequency components after sampling the network and cannot be removed by a low-pass filter (right column).

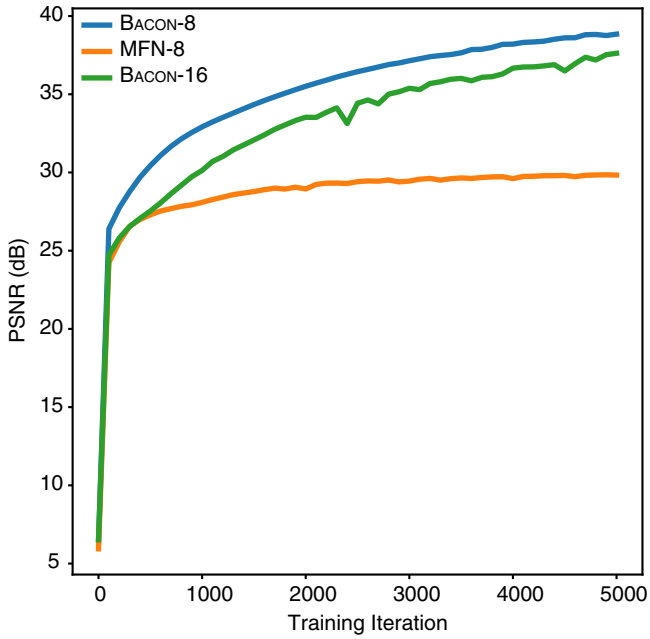


Figure 6. Comparison of deep versions of the original MFN and BACON for image fitting. Both the 8- and 16-layer BACON models fit the lighthouse image to well over 30 dB PSNR while the 8-layer MFN does not reach 30 dB PSNR. We were unable to train a 16-layer MFN with the original initialization scheme due to numerical instabilities during optimization.

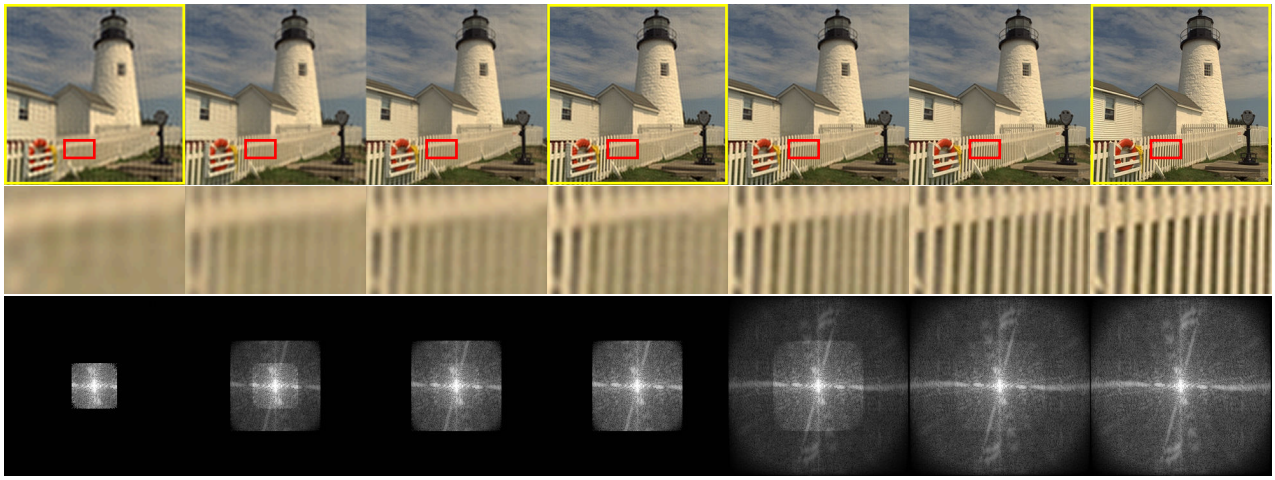


Figure 7. Illustration of interpolation between output scales, similar to the trilinear interpolation used to render from mipmaps. The yellow bordered images in the top row are the outputs of BACON at 1/4, 1/2 and 1× resolution, and the other images are linear interpolations. Zoomed insets are shown in the middle row, and the bottom row shows the Fourier spectrum of each image.

2.2. Neural Radiance Fields

We provide additional implementation details and results on reconstructing neural radiance fields using NeRF [10], Mip-NeRF [3], and BACON.

2.2.1 Additional Implementation Details

For training neural radiance fields, we use BACON with 8 hidden layers and 256 hidden features. We set the bandwidth of each layer using random uniform initialization with $\boldsymbol{\omega}_i \sim \mathcal{U}(-B_i, B_i)$. For a maximum bandwidth B, we set $B_0 = B_1 = B_2 = B/24$, $B_3 = B_4 = B/16$, $B_5 = B_6 = 1/8$, and $B_7 = B_8 = 1/4$ such that $\sum_{i=0}^{8} B_i = B$.

For training BACON we also adopt the regularization strategy of Hedman et al. [9] to penalize non-zero off-surface opacity values, σ. We include this term to mitigate non-zero opacity at unsupervised locations can produce hazy spots in the rendered images using BACON. The regularization penalty is given as

$$\mathcal{L}_{\text{reg}} = \lambda_\sigma \sum_{i,j,k} \log\left(1 + 2\sigma_k(\mathbf{r}_i, \mathbf{t}_j^f)\right), \tag{67}$$

where λ_σ is a weight that we decay from logarithmically from 1e-3 to 1e-6 during training.

To evaluate the effect of the regularization, we train BACON, Mip-NeRF, and NeRF using the same regularizer and report the PSNR in Table 3. All methods are trained on the *lego* scene for 300K iterations with and without regularization. NeRF and Mip-NeRF have few opacity artifacts, and so do not benefit from regularization. BACON shows a significant benefit from regularization.

Additionally, we find that we can obtain a roughly 30% speedup for training and inference without noticeable drop in performance by re-using outputs g_i throughout the network. For example, for layers where $B_i = B_{i+1}$, we set $g_i = g_{i+1}$, allowing us to reuse computation and reducing the number of input layers that need to be computed by roughly half.

scale	NeRF (no reg/reg)	Mip-NeRF (no reg/reg)	BACON (no reg/reg)
1×	**27.695**/27.679	**32.655**/32.436	24.377/**29.658**
1/2	**30.577**/30.565	**33.952**/33.863	24.611/**29.501**
1/4	31.305/**31.311**	**34.058**/34.008	25.105/**29.468**
1/8	**27.067**/27.059	**33.805**/33.762	25.854/**28.958**

Table 3. Evaluation of the effect of opacity regularization for NeRF, Mip-NeRF, and BACON.

2.2.2 Supplemental Results

We provide inference times of NeRF, Mip-NeRF and BACON in Table 4 evaluated on an NVIDIA RTX A6000 GPU. While our implementation is generally slower than the NeRF and Mip-NeRF implementations, we attribute some of this difference to the underlying frameworks; we use PyTorch [11], while NeRF and Mip-NeRF are implemented in JAX [4]. Additionally, the BACON architecture has somewhat greater computational complexity than the comparable NeRF and Mip-NeRF architectures due to the additional sine input layers and Hadamard products. Still, for low-resolution outputs BACON has a computational advantage because only the first few layers need to be evaluated.

In Tables 5 and 6 we provide the per-scene average PSNR and SSIM for each method. We observe the same trends as in the main paper, with Mip-NeRF achieving the best performance, while BACON outperforms NeRF at the lowest and highest resolution outputs and uses a fraction of the parameters to render the low-resolution outputs compared to either baseline. An additional comparison is shown in Table 7 for small versions of the NeRF and Mip-NeRF models trained on the *lego* scene. The number of layers is reduced so that the parameter count is roughly equivalent to the lowest resolution output of BACON. The output PSNR for each method degrades by roughly 1–2 dB at each scale compared to the full-resolution models.

Supplemental qualitative results are shown in Fig. 8, where we show output images for each scene at each scale. Finally, we show additional results for learning neural radiance fields in a semi-supervised case in Fig. 9. Here, outputs of BACON at each scale are supervised on full resolution images, and BACON automatically learns the multiscale decomposition of the neural radiance field used for rendering.

	Inference Times (s)			
	1×	1/2	1/4	1/8
NeRF	4.4	1.1	0.28	0.073
Mip-NeRF	4.5	1.1	0.28	0.073
BACON	10.2	2.1	0.39	0.065

Table 4. Inference times for NeRF, Mip-NeRF, and BACON.

	# Params.	PSNR								
		chair	drums	ficus	hotdog	lego	materials	mic	ship	Avg.
NeRF 1/8		28.767	24.025	25.188	29.685	26.539	24.758	26.720	26.028	26.464
NeRF 1/4		33.064	25.492	26.161	33.478	30.782	26.618	30.615	28.163	29.297
NeRF 1/2	511K	32.882	24.503	25.387	33.711	31.037	25.850	30.517	27.640	28.941
NeRF 1×		29.565	22.741	24.280	31.408	28.228	24.319	27.827	25.508	26.734
NeRF Avg.		31.070	24.190	25.254	32.071	29.147	25.386	28.920	26.834	27.859
Mip-NeRF 1/8		37.174	28.200	28.177	37.332	33.924	30.414	35.803	31.631	32.832
Mip-NeRF 1/4		36.700	26.979	26.951	37.131	34.266	29.233	34.977	30.503	32.093
Mip-NeRF 1/2	511K	35.724	25.560	26.685	36.622	34.295	27.972	34.219	29.379	31.307
Mip-NeRF 1×		33.374	24.005	26.428	34.984	33.136	26.764	32.494	27.808	29.874
Mip-NeRF Avg.		35.743	26.186	27.060	36.517	33.905	28.596	34.373	29.830	**31.526**
BACON 1/8	133K	31.764	25.967	27.184	31.670	29.161	25.899	28.609	27.549	28.475
BACON 1/4	266K	32.523	26.094	25.562	32.175	29.768	25.268	29.524	27.244	28.520
BACON 1/2	398K	31.958	25.074	24.319	32.342	29.890	24.948	29.444	26.552	28.066
BACON 1×	531K	30.729	24.175	23.753	31.942	30.418	24.300	28.454	25.668	27.430
BACON Avg.	329K	31.744	25.327	25.204	32.032	29.809	25.104	29.008	26.753	<u>28.123</u>

Table 5. PSNR for each method averaged over each scene of the multiscale Blender dataset.

	# Params.	SSIM chair	drums	ficus	hotdog	lego	materials	mic	ship	Avg.
NeRF 1/8		0.941	0.902	0.918	0.957	0.930	0.944	0.963	0.876	0.929
NeRF 1/4		0.976	0.926	0.947	0.973	0.968	0.943	0.980	0.888	0.950
NeRF 1/2	511K	0.972	0.909	0.942	0.968	0.961	0.922	0.971	0.865	0.939
NeRF 1×		0.935	0.879	0.925	0.951	0.922	0.895	0.947	0.820	0.909
NeRF Avg.		0.956	0.904	0.933	0.962	0.945	0.926	0.965	0.862	0.932
Mip-NeRF 1/8		0.990	0.952	0.951	0.986	0.984	0.978	0.994	0.928	0.970
Mip-NeRF 1/4		0.989	0.942	0.954	0.983	0.984	0.964	0.990	0.909	0.964
Mip-NeRF 1/2	511K	0.986	0.929	0.960	0.980	0.982	0.949	0.985	0.890	0.957
Mip-NeRF 1×		0.975	0.915	0.957	0.973	0.972	0.932	0.980	0.861	0.946
Mip-NeRF Avg.		0.985	0.935	0.955	0.981	0.980	0.955	0.987	0.897	**0.959**
BACON 1/8	133K	0.962	0.919	0.933	0.967	0.954	0.945	0.970	0.882	0.942
BACON 1/4	266K	0.972	0.931	0.930	0.966	0.948	0.922	0.975	0.877	0.940
BACON 1/2	398K	0.968	0.923	0.927	0.965	0.949	0.913	0.968	0.854	0.934
BACON 1×	531K	0.957	0.917	0.926	0.959	0.951	0.901	0.958	0.827	0.924
BACON Avg.	329K	0.965	0.923	0.929	0.964	0.951	0.921	0.968	0.860	<u>0.935</u>

Table 6. SSIM for each method averaged over each scene of the multiscale Blender dataset.

	# Params.	PSNR 1×	1/2	1/4	1/8	SSIM 1×	1/2	1/4	1/8
NeRF	157K	27.144	30.050	31.554	27.309	0.903	0.949	0.971	0.940
Mip-NeRF	157K	30.136	32.067	32.901	32.798	0.939	0.965	0.977	0.980
BACON	133K	N/A	N/A	N/A	29.161	N/A	N/A	N/A	0.954

Table 7. Comparison between small models trained on the *lego* dataset with roughly equal numbers of parameters as the lowest resolution output of BACON. Reducing the parameter count of NeRF and Mip-NeRF results in a roughly 1–2 dB loss at each scale output.

Figure 8. Neural radiance field results. Outputs of NeRF [10], Mip-NeRF [3], and BACON are shown, where all outputs are supervised using each scale of the multiscale Blender dataset.

Figure 9. Results of training BACON with all outputs supervised at high resolution. BACON learns a multiscale decomposition for each scene. Fourier spectra of the learned opacity volume are shown as insets.

2.3. 3D Shape Representation

2.3.1 Additional Implementation Details

We evaluate BACON on 3D shape representation with SDFs and compare its performance to three other methods: Fourier Features [14], SIREN [12], and Neural Geometric Level of Detail (NGLOD) [13]. All networks are trained to directly fit a signed distance function estimated from a ground-truth mesh. For BACON, Fourier Features, and SIREN we use 8 hidden layers with 256 hidden features. We scale the shape models so that they fit within a volume whose dimensions extend from -0.5 to 0.5. The maximum bandwidth of BACON is set to 256 cycles per unit interval for the *Lucy* and *Thai Statue* scenes, and to 192 cycle per unit interval for the *Armadillo*, *Dragon*, and *Sphere* models. We scale the bandwidth of each layer the same as with the neural radiance field models as described in the main text. We determine the maximum bandwidth empirically such that the network achieves a good fit to high frequency details in the model with noticeable smoothing in the lower levels of detail.

After training, the models are extracted at 512^3 resolution using marching cubes, and we evaluate performance using Chamfer distance and intersection over union (IOU). Chamfer distance is calculated by sampling 300,000 points on the surface of the ground truth and predicted models and then finding the distance to the closest point on the other surface. That is, for the two point clouds P_1, and P_2 we compute

$$D_{\text{Chamfer}}(P_1, P_2) = \frac{1}{|P_1|} \sum_{\mathbf{x} \in P_1} \min_{\mathbf{y} \in P_2} \|\mathbf{x} - \mathbf{y}\|_2^2 + \frac{1}{|P_2|} \sum_{\mathbf{x} \in P_2} \min_{\mathbf{y} \in P_1} \|\mathbf{x} - \mathbf{y}\|_2^2. \tag{68}$$

For the IOU score, we compute intersection and union of the occupancy values for the ground truth and predicted meshes on a 128^3 grid of points centered on the object.

2.3.2 Supplemental Results

We include shape fitting results for four scenes from the Stanford 3D Scanning Repository (*Armadillo*, *Dragon*, *Lucy*, *Thai Statue*) and a simple sphere in Figures 10, 11, 12, 13, and 14. All methods perform similarly at the highest level of detail (see Table 8). For lower levels of detail, BACON (1/8, 1/4, 1/2) represents a smooth multiscale decomposition of the shape, while the representations for NGLOD (NGLOD-1,2,3) show angular artifacts due to their high-frequency spectra (see figure insets). Note that results for NGLOD are shown for training the representation on a maximum of 4 levels of detail (i.e., the number of trained levels of their feature octree) and then rendering out the resulting trained levels of detail 1–4.

Table 8 includes quantitative evaluation of each method for the *Armadillo*, *Dragon*, *Lucy*, *Thai Statue* scenes, and a simple sphere baseline (with radius 0.25). The highest detail outputs of all methods perform comparably, including BACON, which achieves similar performance despite simultaneously representing multiple levels of detail. NGLOD generally improves at higher levels of detail at the cost of significantly more model parameters. Here, NGLOD 1–4 represent outputs from the model trained at maximum level of detail 4, and we also train separate models with maximum levels of detail 5–6 (NGLOD-5 and NGLOD-6).

Additionally, Table 9 includes evaluation of shape fitting at lower levels of detail for BACON and NGLOD (trained with a maximum level of detail 4). Note that here NGLOD has fewer parameters than BACON for lower levels of detail; this is not true for the full resolution models, as number of parameters scales superlinearly for NGLOD and linearly for BACON.

	Parameters	Sphere		Dragon		Armadillo		Lucy		Thai Statue	
		Chamfer↓	IOU↑	Chamfer↓	IOU↑	Chamfer↓	IOU↑	Chamfer↓	IOU↑	Chamfer↓	IOU↑
Fourier Features	527K	8.364e-7	**1.000e+0**	**1.861e-6**	9.828e-1	3.230e-6	9.897e-1	**2.956e-6**	**9.654e-1**	**1.946e-6**	9.823e-1
SIREN	528K	8.407e-7	**1.000e+0**	2.762e-6	9.621e-1	3.895e-6	9.858e-1	3.706e-6	9.625e-1	2.695e-6	9.651e-1
NGLOD-4	1.35M	9.722e-7	9.990e-1	2.272e-6	9.722e-1	3.410e-6	9.891e-1	5.479e-6	9.421e-1	2.320e-6	9.689e-1
NGLOD-5	10.1M	9.443e-7	9.993e-1	2.211e-6	**9.841e-1**	3.804e-6	9.835e-1	3.206e-6	9.621e-1	2.047e-6	9.767e-1
NGLOD-6	78.8M	1.064e-6	9.966e-1	1.918e-6	9.840e-1	**3.212e-6**	**9.911e-1**	3.013e-6	9.634e-1	1.939e-6	**9.824e-1**
BACON 1×	531K	**8.353e-7**	**1.000e+0**	1.875e-6	9.831e-1	3.233e-6	9.893e-1	3.075e-6	9.650e-1	1.972e-6	9.791e-1

Table 8. Quantitative evaluation of 3D shape fitting for high-detail outputs. All methods show comparable performance. Neural Geometric Level of Detail (NGLOD) [13] is shown for levels of detail 4, 5, and 6 and performance generally increases (as does the model parameter count) with increasing levels of detail. BACON gives comparable performance at the highest resolution scale to other methods despite simultaneously representing multiple levels of detail.

	Parameters	Sphere		Dragon		Armadillo		Lucy		Thai Statue	
		Chamfer↓	IOU↑	Chamfer↓	IOU↑	Chamfer↓	IOU↑	Chamfer↓	IOU↑	Chamfer↓	IOU↑
NGLOD-1	8.74K	9.391e-7	9.987e-1	6.624e-6	9.381e-1	6.435e-6	9.699e-1	1.965e-5	8.936e-1	8.139e-6	9.392e-1
NGLOD-2	36.8K	9.549e-7	9.989e-1	3.247e-6	9.612e-1	4.033e-6	9.839e-1	7.550e-6	9.288e-1	3.474e-6	9.638e-1
NGLOD-3	199K	1.100e-6	9.975e-1	2.274e-6	9.722e-1	3.407e-6	9.891e-1	5.513e-6	9.421e-1	2.325e-6	9.689e-1
BACON 1/8	133K	8.349e-7	**1.000e+0**	3.430e-6	9.624e-1	3.997e-6	9.844e-1	5.309e-6	9.461e-1	3.168e-6	9.622e-1
BACON 1/4	266K	**8.320e-7**	**1.000e+0**	2.223e-6	9.773e-1	3.355e-6	9.892e-1	3.467e-6	9.627e-1	2.119e-6	9.748e-1
BACON 1/2	398K	8.343e-7	**1.000e+0**	**1.955e-6**	**9.815e-1**	**3.242e-6**	**9.897e-1**	**3.174e-6**	**9.652e-1**	**1.985e-6**	**9.799e-1**

Table 9. Quantitative evaluation of 3D shape fitting for BACON and Neural Geometric Level of Detail (NGLOD) [13] for low levels of detail. Here, NGLOD has fewer parameters than BACON, and BACON generally achieves better performance.

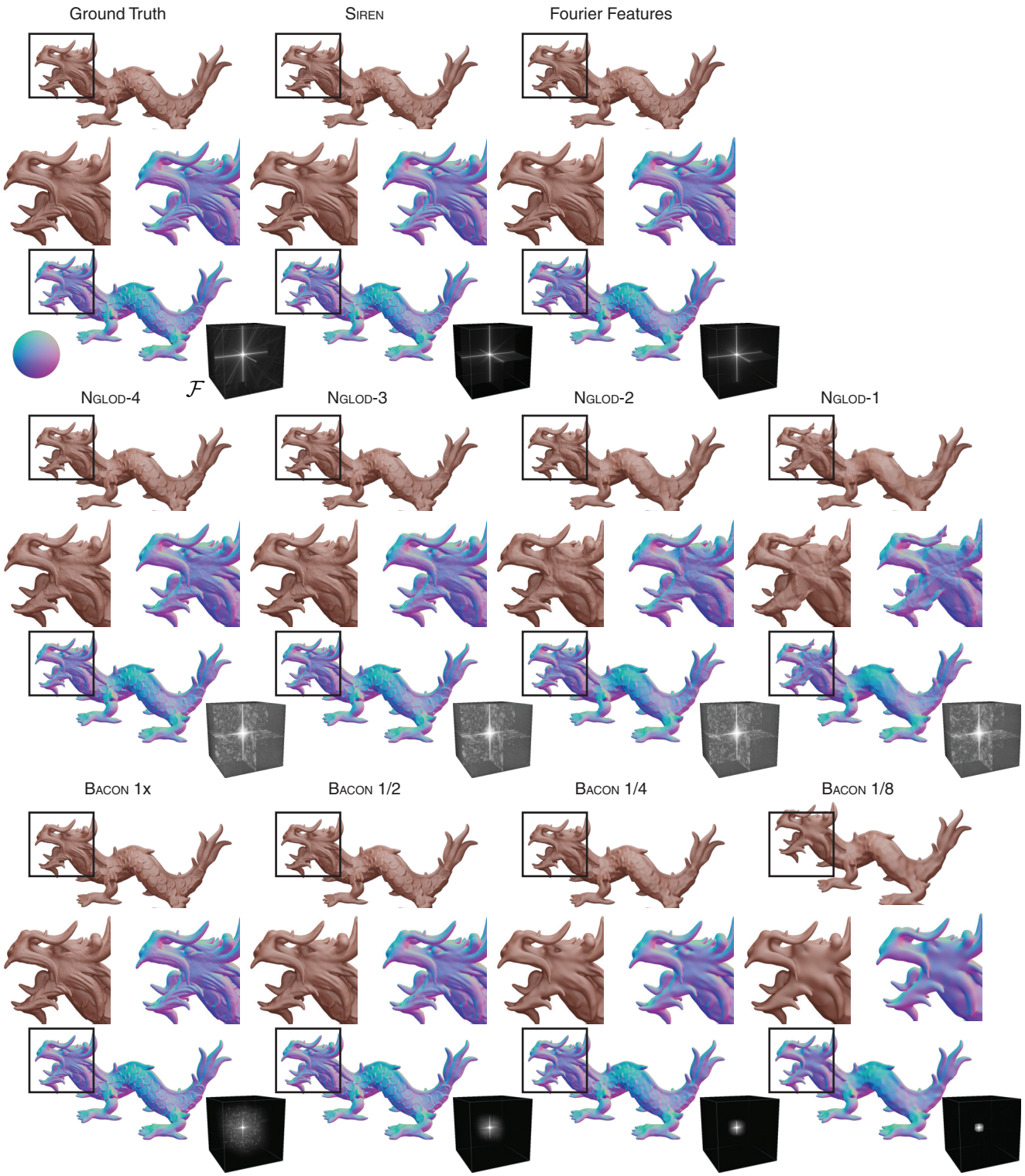


Figure 10. Qualitative results on the *Dragon* scene. Rendered objects and normal maps are shown, and Fourier spectra of the SDF values are included as insets.

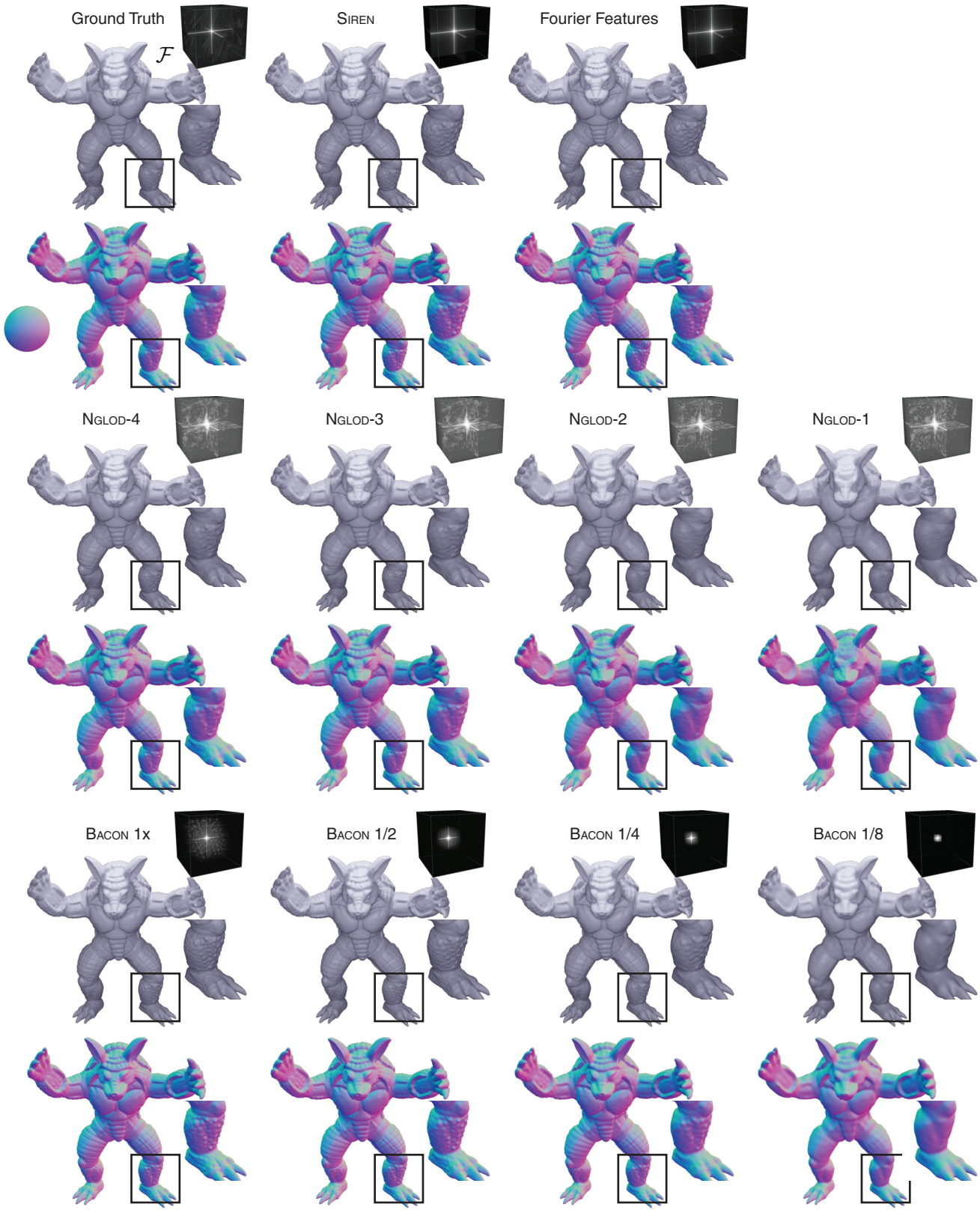


Figure 11. Qualitative results on the *Armadillo* scene. Rendered objects and normal maps are shown, and Fourier spectra of the SDF values are included as insets.

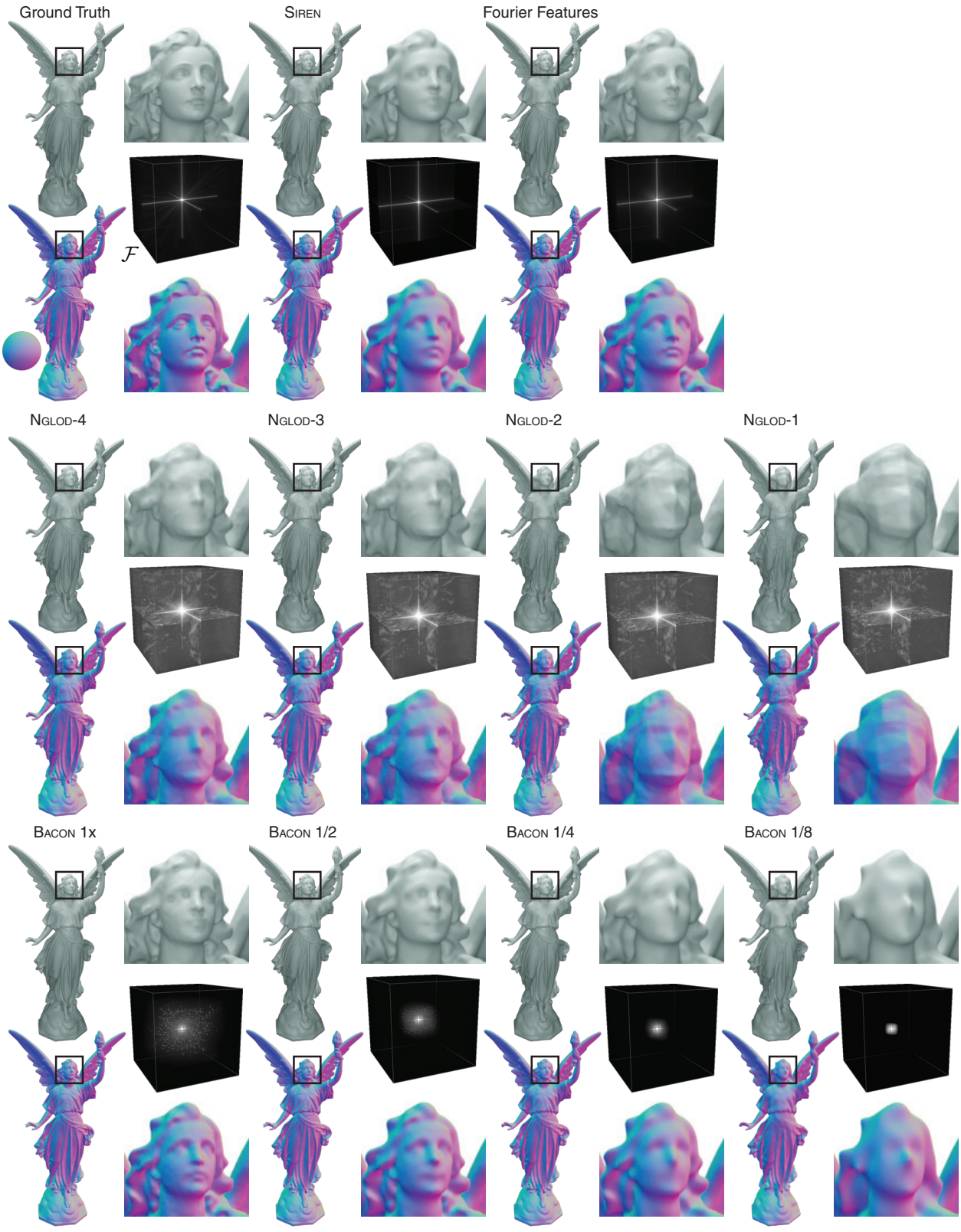


Figure 12. Qualitative results on the *Lucy* scene. Rendered objects and normal maps are shown, and Fourier spectra of the SDF values are included as insets.

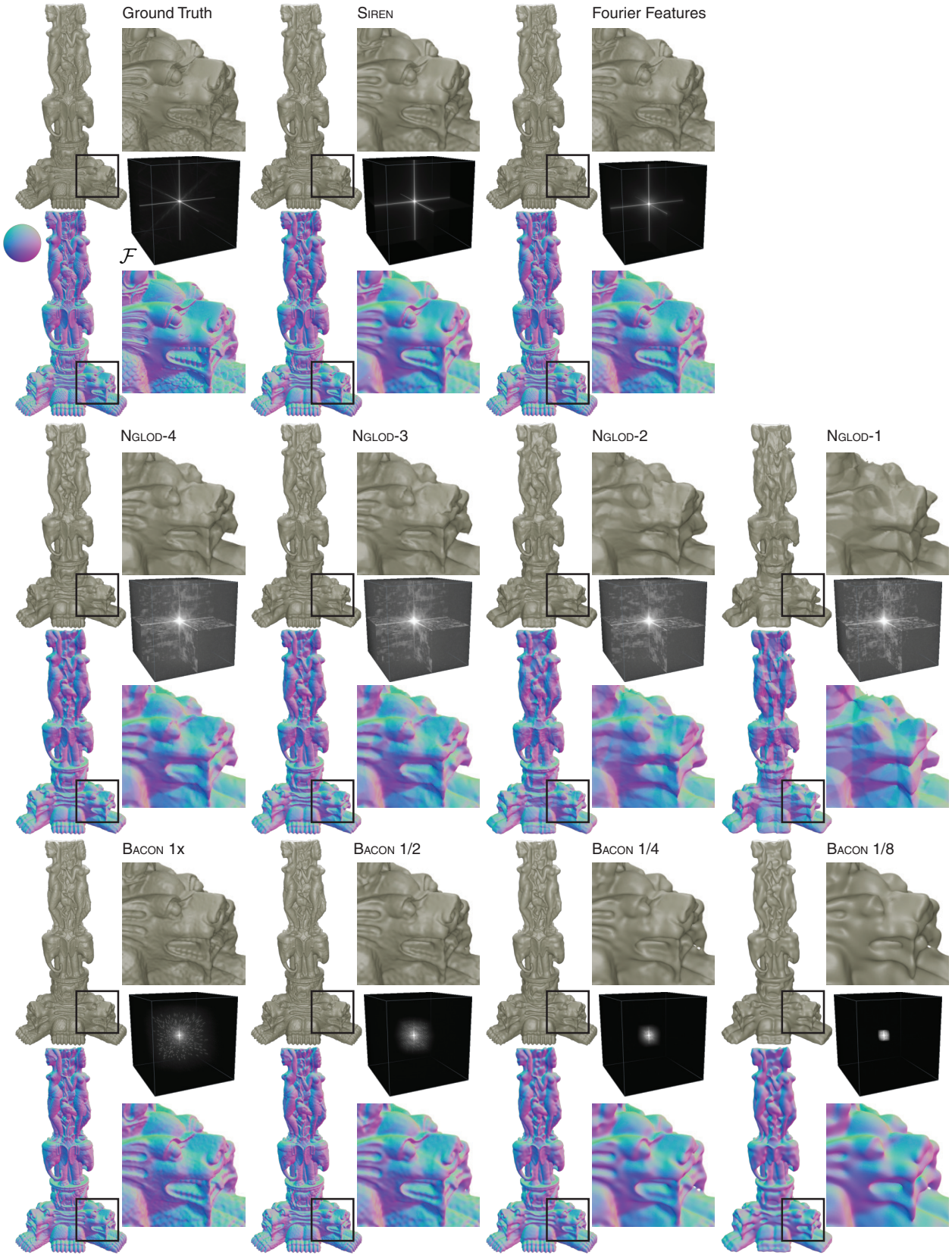


Figure 13. Qualitative results on the *Thai Statue* scene. Rendered objects and normal maps are shown, and Fourier spectra of the SDF values are included as insets.

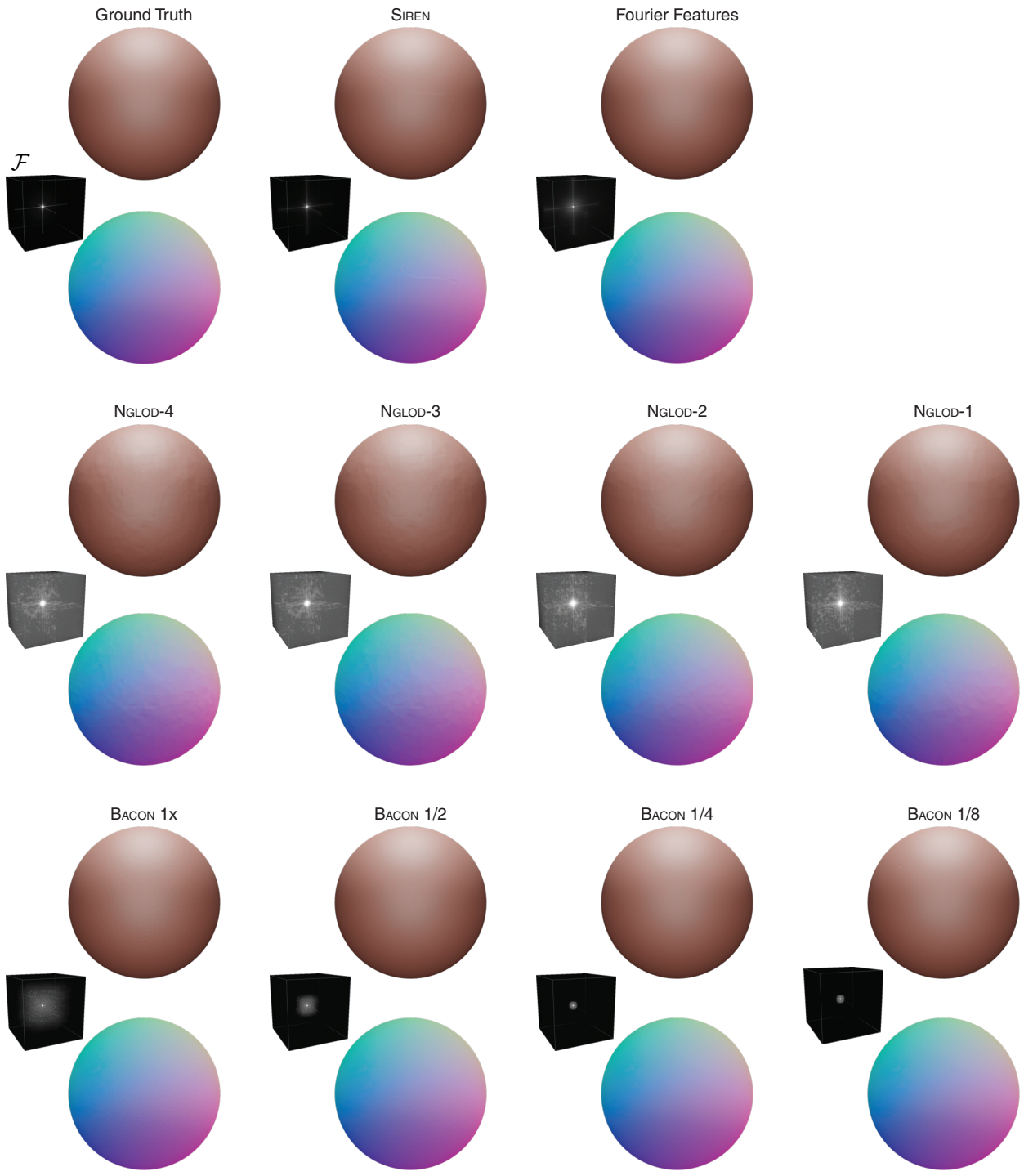


Figure 14. Qualitative results on the *Sphere* scene. Rendered objects and normal maps are shown, and Fourier spectra of the SDF values are included as insets.

2.4. Accelerated Marching Cubes

In this section, we explain two strategies for accelerating mesh extraction via the Marching Cubes algorithm with signed distance function (SDF) representation networks.

2.4.1 Adaptive-Frequency SDF Evaluation

We observe that the band-limited, multi-scale nature of our network allows efficient allocation of computational resources when evaluating SDFs. The key idea is to adaptively choose whether to use SDFs from low-frequency or high-frequency output layers (Fig. 15). For each cell, we compute the low-frequency output SDF_{low} that takes a fraction of time of the full network evaluation. Then, for cells that are far away from the zero-level-set (i.e., magnitude of SDF_{low} larger than some threshold τ), we adopt early-stopping and do not proceed to the higher network layers, as we do not need highly-accurate SDFs for empty cells. For cells near the surface (i.e., $|\text{SDF}_{\text{low}}| < \tau$), we need accurate SDFs, and thus we use the full network for high-frequency outputs. This adaptive early-stopping strategy meaningfully reduces the computation time for mesh extraction (Table 10) and is unique to BACON that produces multi-scale intermediate outputs, unlike the existing architectures such as SIREN that needs to go through the entire network for all cases. We set τ to be 0.7 times the finest voxel length.

2.4.2 Multi-scale SDF Evaluation

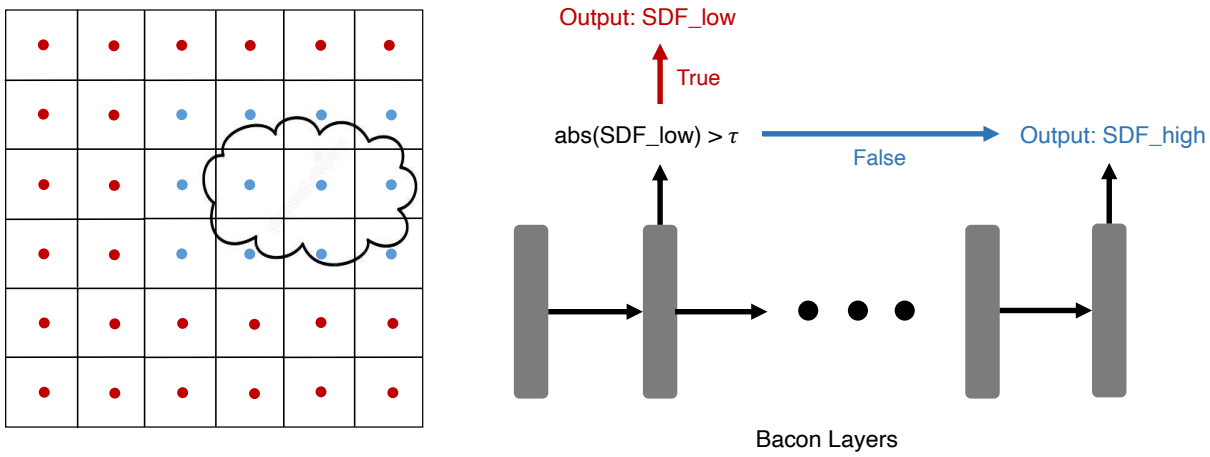
We introduce another strategy to accelerate Marching Cubes mesh extraction using octree-style, multi-scale SDF grids. As shown in Fig. 16, we evaluate the shape SDFs in a hierarchical way, from the low to high resolution grids. We note that the SDFs evaluated at a coarse level can be used to decide whether or not to subdivide a cell. That is, assuming the modeled SDFs are accurate, when the magnitude of SDF at the center of a voxel is larger than the radius of the circumsphere, the voxel is empty (i.e., containing no zero-level-set), so we do not need to further evaluate the SDFs at higher resolutions. Similarly, when the SDF magnitude at the center is smaller than the threshold R, the voxel contains zero-crossing, so it needs to be evaluated at higher resolution via subdivision. Empirically, we set R to be 2 times the circumsphere radius, to provide a margin of safety to the SDF modeling errors. This multi-scale Marching Cubes approach is not unique to BACON and can be applied to other SDF-modeling networks such as SIREN. As shown in Table 10, the strategy reduces the computation time by a factor of $\approx$40.

2.4.3 Combining the Two Strategies

While the above two strategies individually provide significant speedup for mesh extraction, we can combine them together to further enhance the performance. That is, we adopt the adaptive-frequency approach for each level in the multi-scale evaluation. For coarse levels, we adopt the early-stopping strategy to all cells. For the finest resolution level, which takes account for most of the computations, we similarly adopt early-stopping for voxels that are far away from the zero-crossings using the threshold τ. As a result, the combination of the two strategies provide another meaningful reduction of computation time against the pure multi-scale scheme, as shown in Table 10. Note our accelerated Marching Cubes does not decrease the quality of the extracted meshes (see, output shapes in Fig. 17).

2.4.4 Discussion of Occupancy Networks

We notice that Occupancy Networks similarly proposed a multi-resolution mesh extraction strategy on the occupancy fields. The octree-style evaluation for occupancy fields could lead to errors, however, because occupancy fields to not provide the same empty-space guarantees that SDFs provide. Furthermore, the adaptive-frequency evaluation cannot be used for Occupancy Networks, and thus all query points need to be evaluated by the full network layers.



(a) Adaptive-Frequency Outputs

(b) Adaptive-Frequecy Decision Process

Figure 15. Adaptive-frequency SDF evaluations. When evaluating SDFs on a dense grid (a) for mesh extraction, we leverage the band-limited nature of BACON layers to adaptively allocate the computation resources (b) across the cells. For each cell we first compute the SDF with a low-frequency output layer (SDF_{low}). For cells with the magnitude of SDF_{low} larger than some threshold τ (i.e., the red cells that are far from the zero-level-set), we do not proceed to the higher layers of BACON, as we do not need high-frequency details in the empty-space. For cells near the surface (i.e., when $|\text{SDF}_{\text{low}}| < \tau$), we compute the full-frequency SDF (SDF_{high}) using the highest layer output (the blue cells). This adaptive-frequency SDF evaluation saves significant amount of time on computing the SDFs in empty-space via early-stopping, which cannot be adopted by existing network architectures, e.g., SIREN, that need to go through the entire network layers for all evaluations.

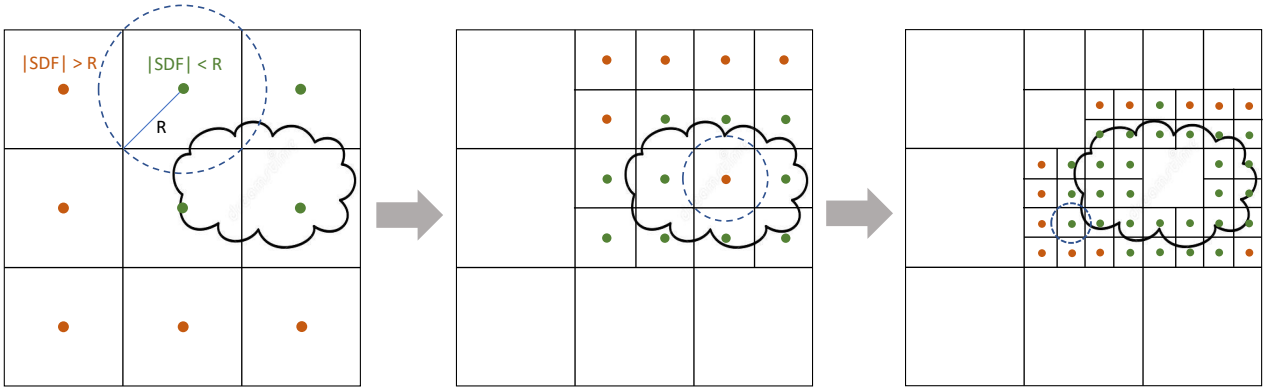


Figure 16. Multi-Scale Marching Cubes. We propose a hierarchical octree-style SDF evaluation scheme for efficiently pruning empty spaces using the nature of SDFs. We start from the coarsest resolution grid (left) and query the SDFs for all cells. Assuming the modeled SDFs are correct, we can identify some of the cells to be empty (i.e, no zero-crossing exists within the cell), when the magnitude of SDFs at the center is larger than the the radius of the circumsphere (dotted circles in the diagram). In practice, to provide a margin of safety for the model errors, we use $2 \times R$ for the criteria for checking empty cells. Then, only for the non-empty cells (green), we subdivide them into 8 cells and evaluate higher resolution SDFs in the next level grid, which we repeat multiple times. For all our experiments we used 4 levels of scales. Note that the proposed multi-scale SDF evaluation is not unique to BACON and can be applied to existing networks, e.g., SIREN, that model SDFs.

	Seconds				
	Armadillo	Dragon	Lucy	Sphere	Thai
SIREN Original	16.75	16.78	16.76	16.78	16.76
SIREN Multi-Scale	0.404	0.258	0.253	0.252	0.354
BACON Original	17.93	17.836	17.909	17.926	17.938
BACON Adaptive	5.925	5.325	5.278	5.391	5.584
BACON Multi-Scale	0.411	0.273	0.270	0.265	0.364
BACON Adapt. + Multi. (Proposed)	**0.280**	**0.207**	**0.188**	**0.172**	**0.267**

Table 10. Marching Cubes timing analysis. From top to bottom: dense vanilla Marching Cubes using SIREN; multi-scale Marching Cubes using SIREN; dense vanilla Marching Cubes using BACON; adaptive mesh extraction using BACON; multi-scale Marching Cubes using BACON; the proposed combination of multi-scale and adaptive SDF evaluation using BACON.

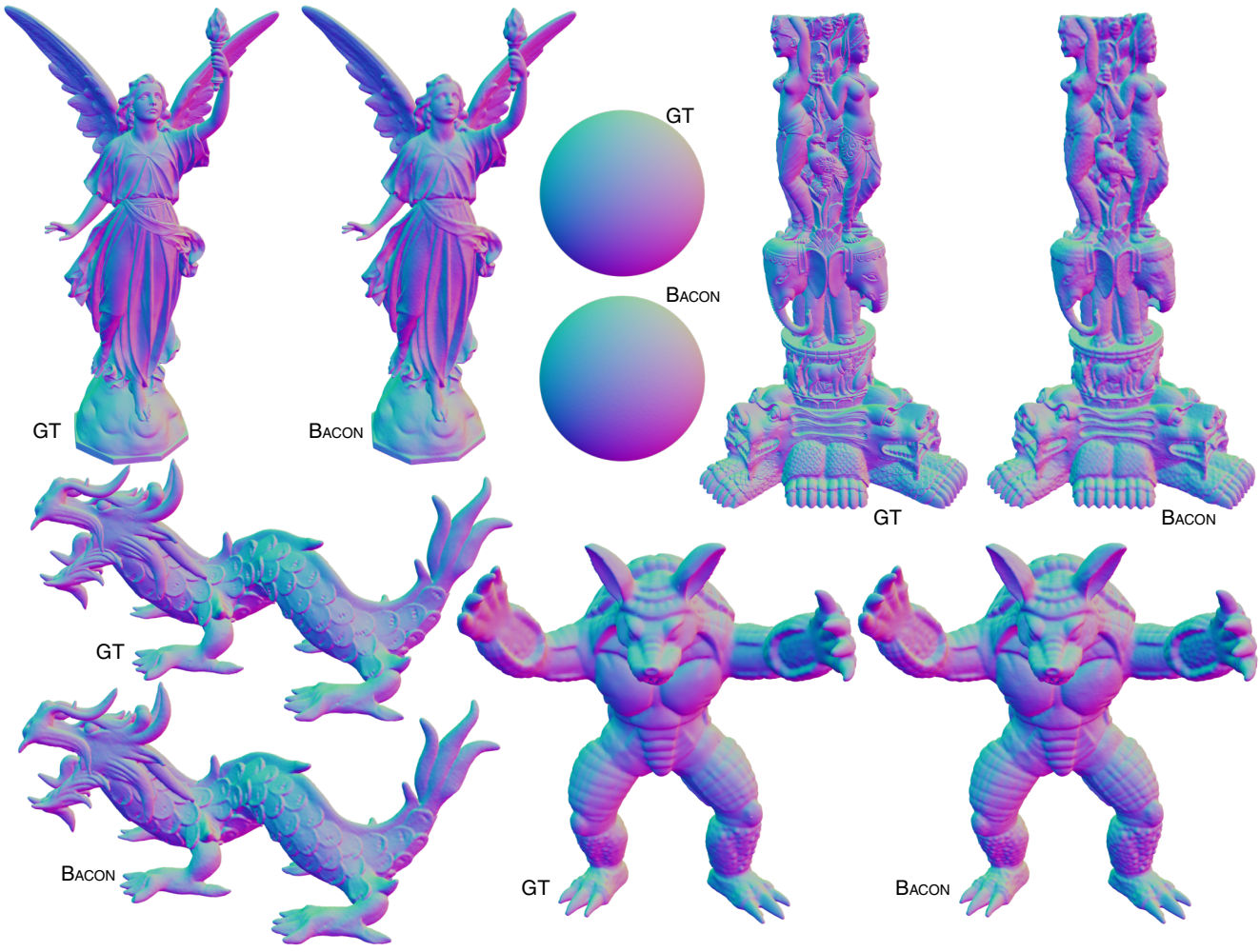


Figure 17. Extracted 3D shapes using the proposed adaptive-frequency multiscale inference procedure.

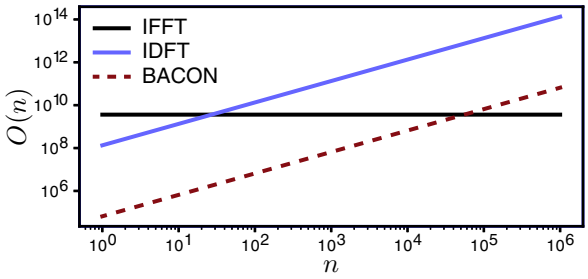


Figure 18. Comparison of asymptotic computational complexity.

2.5. Comparison to Explicit Fourier Basis

Interestingly, we find that a BACON representing a grid of 512^3 discrete frequencies (used in our shape fitting experiments) is more efficient to evaluate for few samples than using the Inverse Fast Fourier Transform (IFFT) or naive computation of the inverse discrete Fourier transform (IDFT) on an explicit grid of 512^3 coefficients. The computational complexity of the IFFT and IDFT are $O(N \log(N))$ and $O(N^2)$, where here, $N = 512^3$. BACON is a compressive representation of the spectrum, and its complexity scales as $O(d_{\text{h}}^2)$ (due to matrix multiplication). In Fig. 18 we plot a comparison of asymptotic computational complexity for n output samples for each of these methods. Since the IFFT always computes 512^3 outputs, its cost is constant.

References

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *CVPR Workshops*, 2017. 10

[2] Robert B Ash, B Robert, Catherine A Doleans-Dade, and A Catherine. *Probability and measure theory*. Academic Press, 2000. 4

[3] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *Proc. ICCV*, 2021. 10, 12, 16, 19

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 16

[5] C. Chatfield and C. M. Theobald. Mixtures and random sums. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 22(4):281–287, 1973. 4

[6] Rizal Fathony, Anit Kumar Sahu, Devin Willmott, and J Zico Kolter. Multiplicative filter networks. In *Proc. ICLR*, 2020. 8, 9

[7] Leo A Goodman. On the exact variance of products. *Journal of the American statistical association*, 55(292):708–713, 1960. 7

[8] Geoffrey Grimmett and David Stirzaker. *Probability and random processes*. Oxford university press, 2020. 6

[9] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *ICCV*, 2021. 16

[10] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proc. ECCV*, 2020. 16, 19

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, et al. Pytorch: An imperative style, high-performance deep learning library. In *Proc. NeurIPS*, 2019. 16

[12] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020. 9, 10, 12, 21

[13] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *Proc. CVPR*, 2021. 21, 22

[14] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Proc. NeurIPS*, 2020. 10, 12, 21

[15] Lance Williams. Pyramidal parametrics. *Computer Graphics (Proc. SIGGRAPH)*, 17(3):1–11, 1983. 10