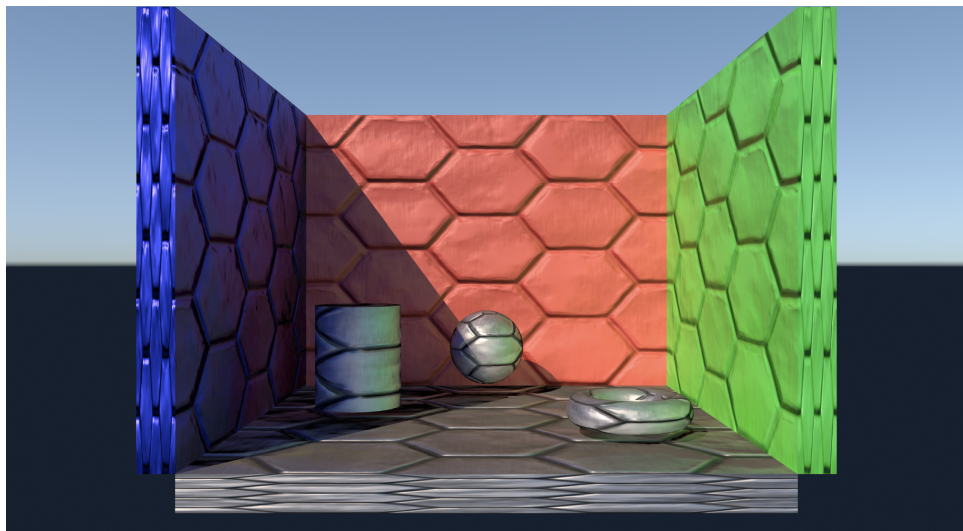


# Progressive Least-Squares Encoding for Linear Bases

Thomas Roughton  
Wellington, New Zealand



**Figure 1.** Indirect lighting represented through a progressively-baked nonnegative lightmap. (Scene credit The Baking Lab [[Pettineo 2018](#)].)

## Abstract

Linear basis functions can be used to encode spherical functions in a compressed format, wherein information such as a radiance field may be represented by a fixed set of basis functions and corresponding basis coefficients. In computer graphics, the function to encode is often generated by way of Monte-Carlo integration, and, in contexts such as lightmap or irradiance volume baking, it is useful to display a progressive result.

This paper presents an efficient, easily-implemented, GPU-compatible method for progressively performing approximate least-squares encoding into arbitrary linear bases (Listing 1). The method additionally supports approximate nonnegative encoding, ensuring that the reconstructed function is positive-valued and improving appearance in a range of scenarios.

```
// The acceleration factor  $\alpha$ .
// A value between 1 and 5 is reasonable.
const float acceleration = 1.0;

Colour coefficients[functionCount] = 0.0;
float mcIntegrals[functionCount] = 0.0;
float totalSampleWeight = 0.0;

func updateEstimate(sample) {
    totalSampleWeight += sample.weight;
    float sampleWeightScale = 1.0 / totalSampleWeight;
    Colour delta = sample.value;
    float sampleLobeWeights[functionCount];

    for functionIndex in 0..<functionCount {
        float weight = evaluateBasisFunction(functionIndex,
                                             sample.direction);
        delta -= coefficients[functionIndex] * weight;
        sampleLobeWeights[functionIndex] = weight;
    }

    for functionIndex in 0..<functionCount {
        float weight = sampleLobeWeights[functionIndex];
        float integralGuess = weight * weight;
        mcIntegrals[i] += (integralGuess
                          - mcIntegrals[i])
                          * sampleWeightScale;

        float basisIntegral = sampleWeightScale
                               + (1.0 - sampleWeightScale)
                               * mcIntegrals[i];

        float deltaScale = acceleration * weight
                             * sampleWeightScale / basisIntegral;
        coefficients[functionIndex] += delta * deltaScale;

        if nonNegativeSolve {
            coefficients[functionIndex] = max(
                coefficients[functionIndex],
                Colour(0)
            );
        }

        if gaussSeidelIteration {
            delta *= 1.0 - deltaScale * weight;
        }
    }
}
```

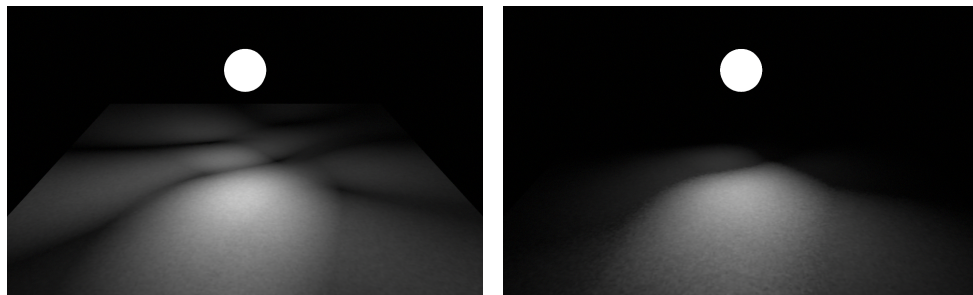
Listing 1. Progressive Least-Squares Encoding.

## 1. Introduction

Linear bases have long been popular in computer graphics as a means of storing encoded radiance information in a compact format. The most common basis functions used come from the spherical harmonics, a family of orthonormal basis functions that can be efficiently encoded, decoded, and convolved in real time. However, spherical harmonics are far from the only useful set of spherical basis functions; the Ambient Cube [Mitchell et al. 2006], Ambient Dice [Iwanicki and Sloan 2017], Ambient Highlight Direction [Sloan and Silvennoinen 2018], and spherical Gaussian [Wang et al. 2009] basis functions have all seen use in real-time applications. These formats can provide advantages in real-time reconstruction; for example, they can have reduced bandwidth requirements or memory footprints compared to high-order spherical harmonics, and some can be used to more accurately approximate specular irradiance. However, these formats are nonorthonormal and are therefore costly to encode; encoding in a least-squares manner to minimize the error in the approximation requires multiplication by an  $N \times N$  matrix, where  $N$  is the number of basis functions, either as the final step in the encoding or per-sample.

Progressive encoding, in which the encoded result may be used immediately as each sample is produced, is becoming increasingly important for interactive artist workflows. For example, interactive lightmap baking tools require progressive visualization of the baked lighting (Figure 1), and for nonorthonormal basis functions the per-sample matrix multiplication necessary imposes significant computational cost.

Standard least-squares solves additionally impose no constraints upon the basis coefficients. In recent years, functions such as spherical Gaussians have been used to represent light sources, where each basis function and its coefficient form a lobe that can be evaluated for irradiance according to some BRDF [Neubelt and Pettineo 2017]. If the basis coefficients are allowed to be negative, this introduces negative light, which is physically implausible and produces visual artifacts (Figure 2). Nonnegative solves avoid this issue, but in prior work could not be performed efficiently on the GPU and required the full sample set.



(a) Least squares

(b) Nonnegative least squares

**Figure 2.** Indirect specular from spherical Gaussian lightmaps.

In this paper, we propose a novel method for progressive least-squares encoding for spherical basis functions. This method is efficient, can run on the GPU, converges fairly quickly, and requires storing only the current amplitude  $b_i$  and a weight for each basis coefficient. Crucially, it allows the imposition of arbitrary constraints upon the amplitudes by simply projecting the constraints onto the values at each iteration of the algorithm.

## 2. Background

When representing a function in a linear basis comprised of a set of basis functions  $B_i(s)$ , the goal is to find the weight vector  $b$  that minimizes the error between the target function  $f(s)$  and the approximation  $\sum_i b_i B_i(s)$ . Since, in general,  $f(s)$  may not be exactly represented in the linear basis, we instead need to minimize the error according to some metric.

One such metric is the least-squares error, defined as the squared difference between the true value and the approximation. The least-squares error may be easily solved for and provides good quality results, although it does disproportionately weight outliers. Minimizing the least-squares error can be done in a functional-analysis manner by solving the following equation:

$$\min \int_S \left( \sum_i b_i B_i(s) - f(s) \right)^2 ds,$$

where  $S$  is the integration domain; typically, for radiance fields, this will be over the sphere, but the same techniques apply over the hemisphere or over other arbitrary domains.

To minimize, we differentiate the function with respect to each unknown  $b_i$  and then set the derivative to 0:

$$E = \int_S \left( \sum_i b_i B_i(s) - f(s) \right)^2 ds;$$

$$\frac{dE}{db_i} = 0.$$

Let  $g(s) = \sum_j b_j B_j(s) - f(s)$ .  $\frac{d}{db_j} [g(s)] = B_j(s)$  for each  $b_j$ :

$$\begin{aligned} \frac{dE}{db_i} &= \frac{d}{db_i} \left[ \int_S \left( \sum_i b_i B_i(s) - f(s) \right)^2 ds \right] \\ &= \frac{d}{db_i} \left[ \int_S (g(s))^2 ds \right] \\ &= 2 \int_S g(s) \frac{d}{db_i} [g(s)] ds \end{aligned}$$



$$\begin{aligned}
 &= 2 \int_S g(s) B_i(s) \, ds \\
 &= 2 \left( \sum_j b_j \int_S (B_i(s) \cdot B_j(s)) \, ds - 2 \int_S (B_i(s) \cdot f(s)) \, ds \right).
 \end{aligned}$$

Therefore, by setting  $\frac{dE}{db_i} = 0$ , we have

$$\sum_j b_j \int_S (B_i(s) \cdot B_j(s)) \, ds = \int_S (B_i(s) \cdot f(s)) \, ds. \quad (1)$$

On the right-hand side we have the *raw moments*, which, when performing Monte-Carlo integration, are the projection of the sample values onto the basis functions;<sup>1</sup> on the left, we have the weight vector  $b$  multiplied by the Gram matrix, where  $G_{ij} = \int_S (B_i(s) \cdot B_j(s)) \, ds$ . To find the weight vector  $b$ , we can multiply the raw moments by the inverse of the Gram matrix:<sup>2</sup>

$$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{pmatrix} \int_S (B_1(s) \cdot B_1(s)) & \int_S (B_1(s) \cdot B_2(s)) & \dots & \int_S (B_1(s) \cdot B_n(s)) \\ \int_S (B_2(s) \cdot B_1(s)) & \int_S (B_2(s) \cdot B_2(s)) & \dots & \int_S (B_2(s) \cdot B_n(s)) \\ \vdots & \vdots & \ddots & \vdots \\ \int_S (B_n(s) \cdot B_1(s)) & \int_S (B_n(s) \cdot B_2(s)) & \dots & \int_S (B_n(s) \cdot B_n(s)) \end{pmatrix}^{-1} \begin{bmatrix} \int_S (B_1(s) \cdot f(s)) \\ \int_S (B_2(s) \cdot f(s)) \\ \vdots \\ \int_S (B_n(s) \cdot f(s)) \end{bmatrix}.$$

Note that, in the case of orthogonal basis functions, the Gram matrix  $G$  will be a diagonal matrix, and for orthonormal basis functions (such as spherical harmonics) the Gram matrix will be the identity matrix.

If the raw moments are computed by Monte-Carlo integration, the Gram matrix can be applied to each sample to perform progressive encoding. Given a set of samples where the  $k$ th sample  $s_k$  has the value  $f(s_k)$ , the weight vector  $b$  is given by

$$b = \frac{1}{k} \sum_k f(s_k) (G^{-1} B(s_k)).$$

## 2.1. Jacobi and Gauss-Seidel Iteration

Jacobi and Gauss-Seidel iteration are two iterative algorithms for solving systems of linear equations in a least squares manner. At each step, Jacobi iteration applies the

<sup>1</sup>Projecting the sample values onto the basis functions simply means averaging the result of multiplying each sample value  $f(s)$  by each basis function  $B_i(s)$ .

<sup>2</sup>The integration variable  $ds$  is here omitted for compactness.

following method to solve the equation  $Ax = b$ , where  $A$  is a matrix and  $x$  and  $b$  are vectors:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} A_{ij} x_j^{(k)}}{A_{ii}}.$$

Gauss-Seidel iteration differs only in that it updates each element of the  $x$  vector in turn and uses the updated elements to calculate the rest; Jacobi iteration will compute the entirety of the  $x^{(k+1)}$  vector before overwriting any part of the previous  $x^{(k)}$  vector.

In their traditional forms, neither Jacobi nor Gauss-Seidel is particularly useful for progressive least-squares encoding in linear bases. If applied to the full system variant, we are required to have all samples in advance in the form of accumulated moments, making the process nonprogressive; alternatively, a system that minimizes the error with regard to each sample produces an unusably noisy estimate to the  $b$  vector.

Both Jacobi and Gauss-Seidel iteration only conditionally converge. Jacobi iteration is known to converge when the system is diagonally dominant; in the case of least-squares projection, this means

$$\int_S B_i(s)^2 ds > \sum_{j, j \neq i} \left| \int_S B_i(s) B_j(s) ds \right|$$

for all  $i$ , although it may also converge under other conditions. Gauss-Seidel iteration, on the other hand, will converge in any case where the Gram matrix is symmetric and positive definite, which will always be the case if the basis functions are strictly positive-valued. Under both Jacobi and Gauss-Seidel iteration, a more diagonally dominant matrix will converge more quickly than a less diagonally dominant one.

### 3. Progressive Least Squares

The conceptual underpinning of the progressive least-squares method is to try to evaluate how accurately  $\sum_i b_i B_i(s)$  approximates each incoming sample  $f(s)$  and to adjust each basis coefficient  $b_i$  by the difference in a form of gradient descent. More formally, the method is a special case of Jacobi or Gauss-Seidel iteration for when the function space is iteratively sampled.

To derive the method, we start with Equation (1) and solve for a single  $b_i$ , assuming that all  $b_j$  are known from a prior iteration:

$$\begin{aligned} \int_S (B_i(s) \cdot f(s)) ds &= \sum_j b_j \int_S (B_i(s) \cdot B_j(s)) ds \\ &= b_i \int_S B_i(s)^2 ds + \sum_{j, j \neq i} b_j \int_S (B_i(s) \cdot B_j(s)) ds. \end{aligned}$$

Then,

$$b_i \int_S B_i(s)^2 ds = \int_S (B_i(s) \cdot f(s)) ds - \sum_{j,j \neq i} b_j \int_S (B_i(s) \cdot B_j(s)) ds.$$

We can bring the entire right-hand side under the same integral due to the linearity of integration:

$$\begin{aligned} b_i \int_S B_i(s)^2 ds &= \int_S (B_i(s) \cdot f(s) - \sum_{j,j \neq i} b_j (B_i(s) \cdot B_j(s))) ds \\ &= \int_S (B_i(s) \cdot (f(s) - \sum_{j,j \neq i} b_j B_j(s))) ds. \end{aligned}$$

Finally, we end up with the following equation for  $b_i$ :

$$b_i = \frac{\int_S (B_i(s) \cdot (f(s) - \sum_{j,j \neq i} b_j B_j(s))) ds}{\int_S B_i(s)^2 ds}.$$

If we assume an iterative process and use the values from the previous iteration at each step, this becomes

$$b_i^{(k+1)} = \frac{\int_S (B_i(s) \cdot (f(s) - \sum_{j,j \neq i} b_j^{(k)} B_j(s))) ds}{\int_S B_i(s)^2 ds}.$$

There are two integrals here that can be computed iteratively using Monte Carlo integration. The denominator,  $\int_S B_i(s)^2 ds$ , can be precomputed; however, it is more accurate in practice to instead compute the denominator in lockstep with the numerator since doing so helps to cancel out sampling bias. At every step of the algorithm, the denominator is stored separately from the  $b$  vector, requiring for radiance storage  $C + 1$  values per basis function where  $C$  is the number of color channels.

Using this iterative framework, we can simplify the numerator by introducing and factoring out  $b_i^{(k)}$ :

$$\begin{aligned} b_i^{(k+1)} &= \frac{\int_S (B_i(s) \cdot (f(s) - \sum_{j,j \neq i} b_j^{(k)} B_j(s))) ds}{\int_S B_i(s)^2 ds} \\ &= \frac{\int_S (B_i(s) \cdot (f(s) - \sum_j b_j^{(k)} B_j(s) + b_i^{(k)} B_i(s))) ds}{\int_S B_i(s)^2 ds} \\ &= \frac{\int_S (B_i(s) \cdot (f(s) - \sum_j b_j^{(k)} B_j(s))) ds + b_i^{(k)} \int_S B_i(s)^2 ds}{\int_S B_i(s)^2 ds} \\ &\approx \frac{\int_S (B_i(s) \cdot (f(s) - \sum_j b_j^{(k)} B_j(s))) ds}{\int_S B_i(s)^2 ds} + b_i^{(k)}. \end{aligned}$$

Let  $\Delta = f(s) - \sum_j b_j B_j(s)$ , or the difference between the current sample value and the current estimate for the current sample's direction. The value  $\Delta$  is constant for all coefficients within a given a particular sample and therefore only needs to be computed once per iteration. Therefore, for each  $i$ , we can compute the  $b_i$  estimate for a particular sample in direction  $\omega_s$  as

$$\text{Est}_s(b_i^{(k+1)}) = \frac{B_i(\omega_s) \cdot \Delta_s}{\int_S B_i(s)^2 ds} + b_i^{(k)}.$$

Welford's algorithm [Welford 1962] is a numerically stable algorithm for computing the mean and variance of some sample set. At each step, the mean  $\mu$  is updated with a new sample  $s$  as follows:

$$\mu^{(k+1)} = \mu^{(k)} + \frac{s - \mu^{(k)}}{k}.$$

Therefore, to accumulate the various Monte Carlo estimates for  $b$  (which is simply a matter of averaging the estimates for each sample given uniform random sampling), we can apply

$$\begin{aligned} b_i^{(k+1)} &= b_i^{(k)} + \frac{1}{k} \left( \frac{B_i(\omega_s) \cdot \Delta_s}{\int_S B_i(s)^2 ds} + b_i^{(k)} - b_i^{(k)} \right) \\ &= b_i^{(k)} + \frac{1}{k} \left( \frac{B_i(\omega_s) \cdot \Delta_s}{\int_S B_i(s)^2 ds} \right). \end{aligned}$$

This method will iteratively converge to the least-squares solution for  $b$  (Figure 3). The speed of its convergence depends on the sample distribution, the initial estimates, and an acceleration factor  $\alpha$ . It turns out to be possible to increase the convergence rate of the method at the cost of increased visible noise during the solve (since the solution will overshoot and correct itself) by performing, at each step,

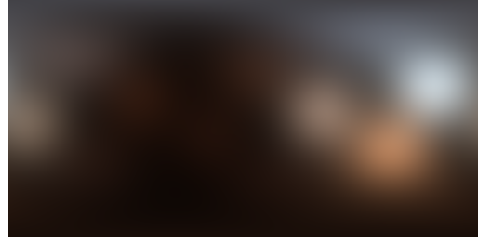
$$b_i^{(k+1)} = b_i^{(k)} + \frac{\alpha}{k} \left( \frac{B_i(\omega_s) \cdot \Delta_s}{\int_S B_i(s)^2 ds} \right).$$

A reasonable range for  $\alpha$  is between 1 and 5. In our tests, we found  $\alpha = 3$  to provide the quickest convergence in a range of scenarios, although the best choice depends on how diagonally dominant the system is and how many samples will be taken.

It is additionally possible to include a per-sample weight if nonuniform sampling or filtered accumulation is being performed using West's extension to Welford's algorithm [West 1979]. Given a weight  $w_s$  for sample  $s$ , the process becomes

$$b_i^{(k+1)} = b_i^{(k)} + \frac{\alpha w_s}{\sum_n w_n} \left( \frac{B_i(\omega_s) \cdot \Delta_s}{\int_S B_i(s)^2 ds} \right).$$

i.e., the samples are multiplied by their weight when accumulating and are averaged by dividing by the total sample weight rather than the sample count.



(a) The Wells HDR environment map [Vogl 2010].

(b) Naive projection onto 12 spherical Gaussian lobes (RMSE: 0.470525).



(c) A least-squares fit with 12 spherical Gaussian coefficients (RMSE: 0.455862).

(d) Progressive least-squares encoding using 12 spherical Gaussian coefficients (RMSE: 0.455971).

**Figure 3.** Comparison of the naive projection, least-squares, and progressive least-squares encoding methods.

### 3.1. Evaluating the Denominator

Special care must be taken in evaluating the denominator  $I = \int_S B_i(s)^2 ds$ . As already mentioned, the denominator should be computed in lockstep with the numerator; as each basis function's weight  $B_i(\omega)$  is evaluated, the value of  $I$  should be updated to be the average of all  $B_i(\omega)^2$  values encountered thus far. However, for low sample counts, the  $b$  estimate will be very noisy, and, since the range of  $B_i^2$  is  $[0, 1]$  for many basis functions, noise in the estimate can be greatly amplified by noise in  $I$ .

We have found two effective methods to mitigate this. The first is to clamp the value used in the denominator for calculating  $b_i$  to at least the precomputed true value of  $I$ . The second method, which has slightly lower error on our test sets, is to interpolate from 1 to the true value based on the sample index  $k$ :

$$I_i^{(k)} = \frac{1}{k} + \left(1 - \frac{1}{k}\right) \cdot \frac{1}{k} \sum_{j=1}^k B_i(\omega_j)^2.$$

### 3.2. Progressive Gauss-Seidel Iteration

The efficacy of this technique is conditional upon the convergence of Jacobi or Gauss-Seidel iteration given a particular basis function: the greater overlap there is between

the basis functions, the slower the system will converge, and if the system is not diagonally dominant it may not converge at all using Jacobi iteration. In these cases, we can instead use a form of Gauss-Seidel iteration by updating the value of  $\Delta$  after solving for every basis coefficient:

$$\begin{aligned}
 \Delta_{i+1}^{(k+1)} &= \Delta_i^{(k+1)} + b_i^{(k)} B_i(\omega) - b_i^{(k+1)} B_i(\omega) \\
 &= \Delta_i^{(k+1)} + (b_i^{(k+1)} - \frac{\alpha}{k} (\frac{B_i(\omega) \cdot \Delta_i^{(k+1)}}{\int_S B_i(s)^2 ds})) B_i(\omega) - b_i^{(k+1)} B_i(\omega) \\
 &= \Delta_i^{(k+1)} - \frac{\alpha}{k} (\frac{B_i(\omega) \cdot \Delta_i^{(k+1)}}{\int_S B_i(s)^2 ds}) B_i(\omega) \\
 &= \Delta_i^{(k+1)} (1 - \frac{\alpha}{k} \frac{B_i(\omega)^2}{\int_S B_i(s)^2 ds}).
 \end{aligned}$$

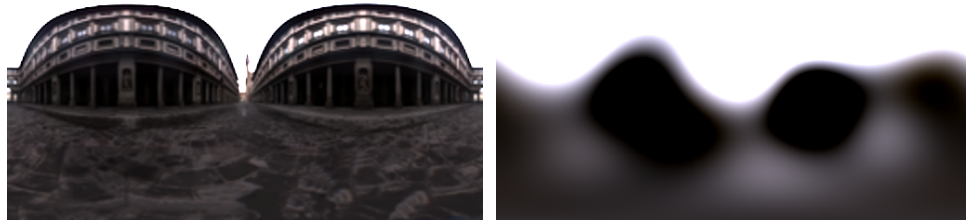
Using Gauss-Seidel iteration, this algorithm will converge to the least-squares solution in the unconstrained case provided that Gauss-Seidel iteration on the full system would converge. As a rough heuristic, this progressive least-squares algorithm exhibits similar error to performing somewhere between five and eight iterations of the Gauss-Seidel algorithm on the full system set up in a functional-analysis least-squares manner (i.e.,  $Gb = m$ , where  $G$  is the Gram matrix,  $b$  is the weight vector, and  $m$  is the vector of projected moments). This is true regardless of whether Gauss-Seidel or Jacobi iteration is used within the algorithm; however, this algorithm will only converge using Jacobi iteration in situations where Jacobi iteration on the full system would converge.

#### 4. Nonnegative Encoding

Extra constraints may be introduced by projecting the basis amplitude onto those constraints after every iteration; for example, a nonnegative solve can be achieved by clamping the amplitude to be nonnegative after each sample is added. Doing so may prevent the system from ever reaching the true value and has no formal mathematical basis, although intuitively you may reason that subsequent samples will compensate for the constraint in their solve. In practice, the results from nonnegative clamping come very close to those achieved using a dedicated nonnegative solver on the full system (Figure 4).

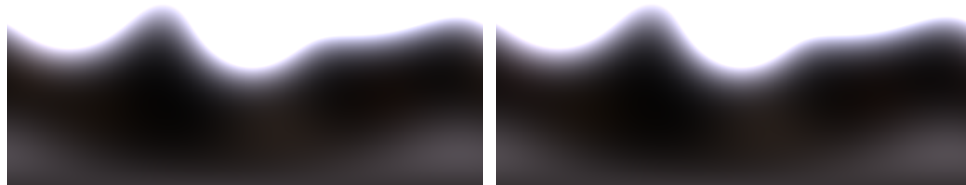
#### 5. Notes and Limitations

The efficacy of this technique is conditional upon the distribution of the incoming samples. If the sample points are distributed across the sampling domain in an uncorrelated or negatively correlated fashion (white noise or blue noise) then the result will converge to the minimum mean-squared error; however, if the sample points are



(a) The Uffizi HDR environment map [Debevec 1998].

(b) A least-squares fit using 12 spherical Gaussian coefficients (RMSE: 3.07549).

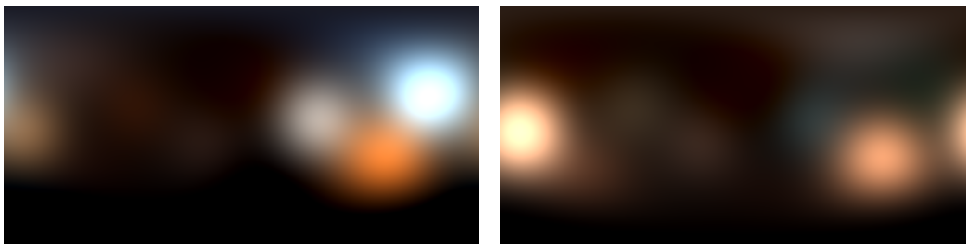


(c) A nonnegative least-squares fit using 12 spherical Gaussian coefficients (RMSE: 3.13181).

(d) Progressive nonnegative least-squares encoding using 12 spherical Gaussian coefficients (RMSE: 3.13928).

**Figure 4.** Progressive least-squares encoding with negative vs. nonnegative lobe amplitudes.

highly correlated the result will be very poor. Fortunately, we naturally want the sampling pattern to be negatively correlated in most contexts where we are accumulating radiance samples progressively; in path tracing, for instance, stratified sampling is often used to ensure that successive samples are not over-representative of a particular direction. Note also that progressive sample *sequences* that converge quickly within the first few samples are ideal, while sample *sets* that cover the sample space only once all samples have been taken are very poor choices (Figure 5).



(a) Progressive least-squares encoding with samples from the Halton 2-3 sequence.

(b) Progressive least-squares encoding with correlated samples from the Hammersley set.

**Figure 5.** Progressive least-squares encoding with correlated vs. decorrelated samples.



If the incoming sample directions are defined uniformly over the hemisphere (as is the case in lightmap baking) but the integration domain is over the sphere, additional samples should be added after each true sample with a direction opposite the upper hemisphere direction and a radiance value of zero.<sup>3</sup> This is particularly important when using fits to approximate some other quantity, such as irradiance from the encoded function; integration over the hemisphere requires clipping of the BRDF by the sampling hemisphere in addition to the BRDF hemisphere and each basis function's domain.

The algorithm is reasonably well-behaved in the presence of noisy input, such as in cases where the function  $f(s)$  is estimated by Monte-Carlo integration. With that said, in lightmap baking contexts, neighbouring texels may reach differing local minima, resulting in a noisier image than a functional-analysis solve would provide (Figure 6).

As outlined in Section 2.1 the convergence rate of this algorithm is largely determined by how diagonally dominant the system is. This means that systems with highly overlapping basis functions may converge slowly and have higher energy in their encoding compared to a system using more orthogonal basis functions (Figure 7).

## 6. Implementation

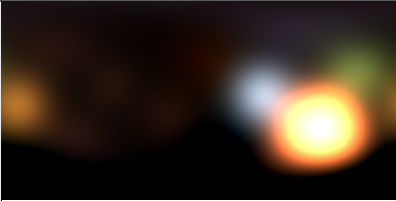
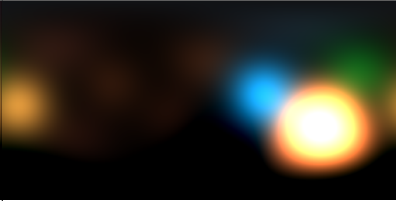
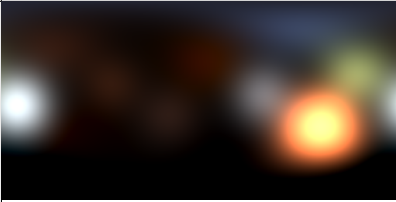
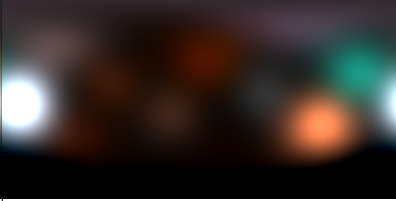
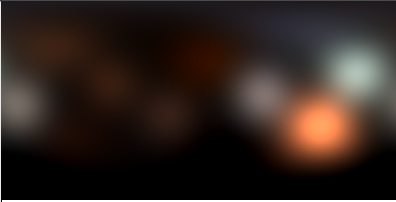
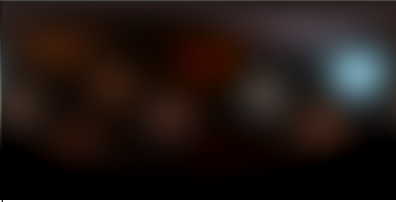

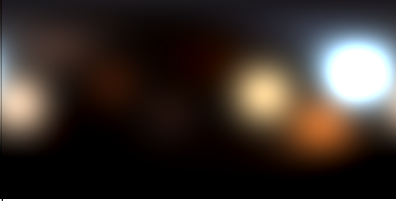
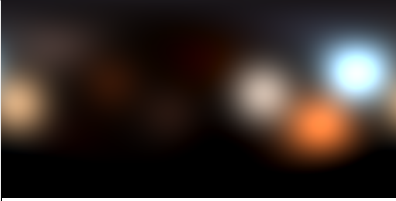
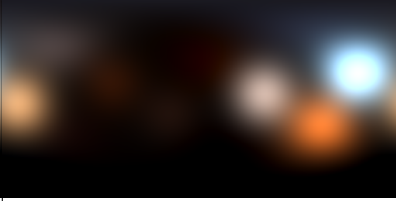
This method has been implemented and tested across a range of different software on both the CPU and GPU. Initial prototyping was done within the open source tool Probulator [O'Donnell 2016], and was later tested within the open source tool The Baking Lab [Pettineo 2018]; the GPU implementation was tested within a closed-source engine.

For the GPU implementation, samples were traced from locations in a lightmap and were accumulated into a 32-bit float RGBA render target per basis function, with the spherical integral  $I$  stored in the alpha channels. A single-channel 32-bit float render target was also used to store the total accumulated sample weight and results were splatted across multiple texels using filtered weights.

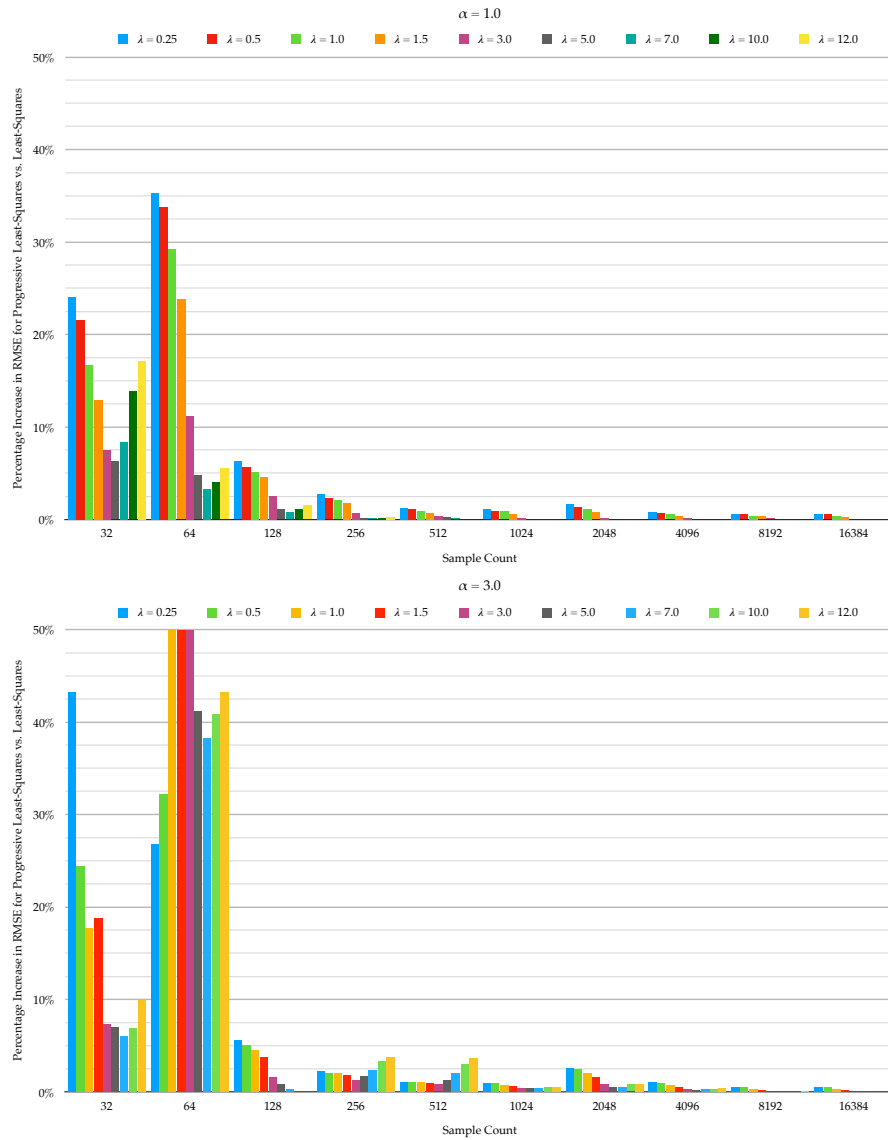
Care must be taken with regard to the storage of the intermediate weight vector  $b$ . In particular, 16-bit floating point is insufficiently precise to capture the minute adjustments to the weights and will cause biasing towards large sample values. In our implementation, all intermediate results were stored in 32-bit floating point; preliminary tests done with 64-bit floating point showed minimal improvement in accuracy over 32-bit.

---

<sup>3</sup>Similar considerations apply when performing least squares by attempting to fit the basis functions evaluated in the sample directions to the sample values directly; doing so when samples are distributed over a hemisphere is an approximation to functional-analysis least squares over the hemisphere.

	Least Squares	Progressive Least Squares
32 Samples		
RMSE	0.566	0.601
64 Samples		
RMSE	0.495	0.507
128 Samples		
RMSE	0.472	0.493
256 Samples		
RMSE	0.462	0.472
512 Samples		
RMSE	0.461	0.461

**Figure 6.** Convergence rate of the progressive least-squares encoding method on the Wells HDR environment map [Vogl 2010] with twelve spherical Gaussian coefficients ( $\lambda = 8$ ).



**Figure 7.** Percentage increase in error vs. sample count for different spherical Gaussian lobe widths ( $\lambda$ , where lower means a wider lobe) and acceleration factors, measured on the Grace HDR environment map [Debevec 1998].

## Acknowledgements

Many thanks to Peter-Pike Sloan for his assistance in understanding the functional-analysis least squares method, and to Matt Pettineo for implementing an earlier version of the algorithm in his open-source tool The Baking Lab.

## Publication Note

An earlier version of this paper was published within the thesis *Interactive Generation of Path-Traced Lightmaps* [Roughton 2019].

## References

- DEBEVEC, P. 1998. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-based Graphics with Global Illumination and High Dynamic Range Photography. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 189–198. doi:10.1145/280814.280864. 27, 30
- IWANICKI, M., AND SLOAN, P.-P. 2017. Ambient Dice. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*, Eurographics Association, Goslar, Germany, 19–29. URL: <https://doi.org/10.2312/sre.20171191>. 19
- MITCHELL, J., MCTAGGART, G., AND GREEN, C. 2006. Shading in Valve's Source Engine. In *ACM SIGGRAPH 2006 Courses*, ACM, New York, NY, USA, SIGGRAPH '06, 129–142. URL: <http://doi.acm.org/10.1145/1185657.1185832>, doi:10.1145/1185657.1185832. 19
- NEUBELT, D., AND PETTINEO, M., 2017. Physically Based Shading in Theory and Practice: Advanced Lighting R&D at Ready At Dawn Studios. Presented at SIGGRAPH 2015. URL: <https://blog.selfshadow.com/publications/s2015-shading-course/>. 19
- O'DONNELL, Y., 2016. Probulator. URL: <https://github.com/kayru/Probulator>. 28
- PETTINEO, M., 2018. The Baking Lab. URL: <https://github.com/TheRealMJP/BakingLab>. 17, 28
- ROUGHTON, T. 2019. *Interactive Generation of Path-Traced Lightmaps*. Master's thesis, Victoria University of Wellington. URL: <https://torust.me/thesis>. 31
- SLOAN, P.-P., AND SILVENNOINEN, A. 2018. Directional lightmap encoding insights. In *SIGGRAPH Asia 2018 Technical Briefs*, ACM, New York, NY, USA, 1–3. URL: <https://doi.org/10.1145/3283254.3283281>, doi:10.1145/3283254.3283281. 19
- VOGL, B., 2010. Light Probes. URL: <http://dativ.at/lightprobes/>. 25, 29
- WANG, J., REN, P., GONG, M., SNYDER, J., AND GUO, B. 2009. All-Frequency Rendering of Dynamic, Spatially-Varying Reflectance. *ACM Trans. Graph.* 28, 5 (Dec.), 133:1133:10. doi:10.1145/1618452.1618479. 19

WELFORD, B. P. 1962. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics* 4, 3, 419-420. URL: <http://www.jstor.org/stable/1266577>. 24

WEST, D. H. D. 1979. Updating mean and variance estimates: An improved method. *Commun. ACM* 22, 9 (Sept.), 532-535. URL: <http://doi.acm.org/10.1145/359146.359153>, doi:10.1145/359146.359153. 24

### Author Contact Information

Thomas Roughton  
[t.rougton@me.com](mailto:t.rougton@me.com)

---

Thomas Roughton, Progressive Least-Squares Encoding for Linear Bases, *Journal of Computer Graphics Techniques (JCGT)*, vol. 9, no. 1, 17-32, 2020  
<http://jcgt.org/published/0009/01/02/>

Received: 2019-05-24

Recommended: 2019-10-09

Published: 2020-01-25

Corresponding Editor: Angelo Pesce

Editor-in-Chief: Marc Olano

© 2020 Thomas Roughton (the Authors).

The Authors provide this document (the Work) under the Creative Commons CC BY-ND 3.0 license available online at <http://creativecommons.org/licenses/by-nd/3.0/>. The Authors further grant permission for reuse of images and text from the first page of the Work, provided that the reuse is for the purpose of promoting and/or summarizing the Work in scholarly venues and that any reuse is accompanied by a scientific citation to the Work.

