Approximating Volume Line Integral by Low-Rank Matrix Approximation

Pramook Khungurn

December 11, 2019

Given a 3D function V represented as a trilinear interpolation of a 3D grid, we would like to construct a data structure that approximates/upper bounds/lower bounds the answer to the following query:

Given two points $\mathbf{p}_0$ and $\mathbf{p}_1$ in space, compute the line integral

$$\|\mathbf{p}_1 - \mathbf{p}_0\| \int_0^1 V(\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)t) \, \mathrm{d}t. \tag{1}$$

1 Motivation and Settings

- Suppose a volume of participating media at point $\mathbf{p}$ has extinction coefficient $\sigma_t(\mathbf{p})$. Then, radiance originating at point $\mathbf{p}_0$ going to point $\mathbf{p}_1$ in the direction of $\mathbf{p}_1 = \mathbf{p}_0$ would be attenuated by a factor of

$$\exp\left(- \|\mathbf{p}_1 - \mathbf{p}_0\| \int_0^1 \sigma_t(\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)t) \, \mathrm{d}t \right). \tag{2}$$

- The attenuation factor (2) is useful for multiple importance sampling and computation of shadow rays.
- Ray marching is a popular technique used in computer graphics to compute the line integral. However, the evaluation is not accurate if the step is too large, and it becomes slow if the step is too small. Moreover, when used in a renderer, the resulting image is biased.
- Simpling techniques such as Woodcock tracking gives a random variable whose expectation is equal to (2), but it cannot evaluate or give an approximation to (2). Hence, the technique cannot be used with multiple importance sampling.
- As such, we seek a data structure or a precomputation scheme that approximates the line integral (1) while incurring small computational cost.
- We focus on cube volumes V that are represented by a 3D grid of size $(n+1) \times (n+1) \times (n+1)$ where $n = 2^m$. The grid partitions the volume into n^3 cells. For each cell, the value of the volume function at a point in the cell is defined to be the trilinear interpolation of the values of the grid points that are the cell's corners.
- In this seeting, we can calculate the line integral through V exactly using an algorithm similar to the 3D differential analyzer. The complexity of evaluating this integral is $O(n)$. We aim to do better than this baseline by sacrificing some quality of the result.

2 Exact Integral for One Cell

- Before we go on to talk about approximations we make, we need to discuss how to compute the exact result.
- Imagine a cube whose bottom corner is at $(0,0,0)$ and whose top corner is at $(1,1,1)$. The volume contained in this cube is $[0,1]\times[0,1]\times[0,1]$.

  We view the corners of the grid as a $2\times 2\times 2$ grid. Let c_{ijk}, where $i,j,k\in\{0,1\}$ denote the value stored at the corner (i,j,k).
- Let $V_{i'j'k'}(x,y,z)$ denote a volume function defined on $[0,1]\times[0,1]\times[0,1]$ as the trilinear interpolation of the values stored in following grid:

$$c_{ijk}=\begin{cases}1, & i=i', j=j', \text{ and } k=k',\\ 0, & \text{otherwise}\end{cases}.$$

  We have that

$$\begin{aligned}
V_{000}(x,y,z)&=(1-x)(1-y)(1-z),\\
V_{001}(x,y,z)&=(1-x)(1-y)z,\\
V_{010}(x,y,z)&=(1-x)y(1-z),\\
V_{011}(x,y,z)&=(1-x)yz,\\
V_{100}(x,y,z)&=x(1-y)(1-z),\\
V_{101}(x,y,z)&=x(1-y)z,\\
V_{110}(x,y,z)&=xy(1-z),\text{ and}\\
V_{111}(x,y,z)&=xyz
\end{aligned}$$

  for all $(x,y,z)\in[0,1]\times[0,1]\times[0,1]$.
- So, for a general cube where values stored at corners can be arbitrary, we let $V(x,y,z)$ denote the volume function defined as the trilinear interpolation of the values stored at the corners. We have that

$$V(x,y,z)=c_{000}V_{000}(x,y,z)+c_{001}V_{001}(x,y,z)+\cdots+c_{111}V_{111}(x,y,z).$$

- Given two points $\mathbf{p}_0=(x_0,y_0,z_0)$ and $\mathbf{p}_1=(x_1,y_1,z_1)$ inside the volume $[0,1]\times[0,1]\times[0,1]$, the line connecting them is given by the function $\mathbf{p}(t)=\mathbf{p}_0+(\mathbf{p}_1-\mathbf{p}_1)t$.
- Again, we are interested in computing the line integral.

$$\mathcal{L}_V(\mathbf{p}_0,\mathbf{p}_1)=\|\mathbf{p}_1-\mathbf{p}_0\|\int_0^1 V(\mathbf{p}_0-(\mathbf{p_1}-\mathbf{p_0})t)\ \mathrm{d}t.$$

  We have that

$$\begin{aligned}
\mathcal{L}_V(\mathbf{p}_0,\mathbf{p}_1)&=\|\mathbf{p}_1-\mathbf{p}_0\|\int_0^1 V(\mathbf{p}(t))\ \mathrm{d}t\\
&=\|\mathbf{p}_1-\mathbf{p}_0\|\int_0^1\Big(c_{000}V_{000}(\mathbf{p}(t))+\cdots c_{111}V_{111}(\mathbf{p}(t))\Big)\ \mathrm{d}t\\
&=c_{000}\|\mathbf{p}_1-\mathbf{p}_0\|\int_0^1 V_{000}(\mathbf{p}(t))\mathrm{d}t+\cdots+c_{111}\|\mathbf{p}_1-\mathbf{p}_0\|\int_0^1 V_{111}(\mathbf{p}(t))\mathrm{d}t\\
&=c_{000}\mathcal{L}_{V_{000}}(\mathbf{p}_0,\mathbf{p}_1)+\cdots+c_{111}\mathcal{L}_{V_{111}}(\mathbf{p}_0,\mathbf{p}_1).
\end{aligned}$$

  Hence, $\mathcal{L}_{V_{000}}$, $\mathcal{L}_{V_{001}}$, …, and $\mathcal{L}_{V_{111}}$ are basis functions of trilinearly interpolated volume.

- We can evaluate the integrals using the following identity:

$$\begin{aligned}\int_0^1 (at+b)(ct+d)(et+f)\ \mathrm{d}t &= \int_0^1 \big(acet^3 + (acf+ade+bce)t^2 + (adf+bcf+bde)t + bdf\big)\ \mathrm{d}t \\ &= \frac{ace}{4} + \frac{acf+ade+bce}{3} + \frac{adf+bcf+bde}{2} + bdf.\end{aligned}$$

- Trying the identity out with V_{000}, we have

$$\begin{aligned}\int_0^1 V_{000}(\mathbf{p}(t))\ \mathrm{d}t &= \int_0^1 \big((x_1-x_0)t+x_0\big)\big((y_1-y_0)t+y_0\big)\big((z_1-z_0)t+z_0\big)\ \mathrm{d}t \\ &= \frac{(x_1-x_0)(y_1-y_0)(z_1-z_0)}{4} + \cdots + x_0y_0z_0.\end{aligned}$$

  Using Mathematica to simplify the terms, we have

$$\begin{aligned}\int_0^1 V_{000}(\mathbf{p}(t))\ \mathrm{d}t &= \frac{x_0y_0z_0}{4} + \frac{x_1y_0z_0 + x_0y_1z_0 + x_1y_1z_0 + x_0y_0z_1 + x_1y_0z_1 + x_0y_1z_1}{12} + \frac{x_1y_1z_1}{4} \\ &= \frac{x_0y_0z_0}{6} + \frac{(x_0+x_1)(y_0+y_1)(z_0+z_1)}{12} + \frac{x_1y_1z_1}{6} \\ &= \frac{2x_0y_0z_0 + (x_0+x_1)(y_0+y_1)(z_0+z_1) + 2x_1y_1z_1}{12}. \qquad (3)\end{aligned}$$

- For other V_{ijk}, we can use (3) to evaluate it. For example, for V_{111}, the integrand is

$$\begin{aligned}&\big(1-(x_1-x_0)t-x_0\big)\big(1-(y_1-y_0)t-y_0\big)\big(1-(z_1-z_0)t-z_0\big) \\ &= \big((x_0-x_1)t+1-x_0\big)\big((y_0-y_1)t+1-y_0\big)\big((z_0-z_1)t+1-z_0\big) \\ &= ((x_1'-x_0')t+x_0')((y_1'-y_0')t+y_0')((z_1'-z_0')t+z_0')\end{aligned}$$

  where $x_0' = 1-x_0$, $x_1' = 1-x_1$, and so on. Thus, using (3)

$$\begin{aligned}\int_0^1 V_{111}(\mathbf{p}(t))\ \mathrm{d}t &= \frac{2x_0'y_0'z_0' + (x_0'+x_1')(y_0'+y_1')(z_0'+z_1') + 2x_1'y_1'z_1'}{12} \\ &= \frac{1}{12}\Big(2(1-x_0)(1-y_0)(1-z_0) + (2-x_0-x_1)(2-y_0-y_1)(2-z_0-z_1) \\ &\qquad + 2(1-x_0)(1-y_0)(1-z_0)\Big).\end{aligned}$$

3 A Framework for Multiscale Precomputed Line Integrals

3.1 Notations

- For $k \in \mathbb{Z}^+$, let $\mathbb{Z}_k$ denote the set $\{0, 1, \ldots, k-1\}$.
- We call a function $c : \mathbb{Z}_n \times \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{R}$ a *grid of size n*. For notational convenience, we denote the value of c as c_{ijk} instead of $c(i,j,k)$.
- Let c be a grid of size n. We will discuss *subgrids* of c, and we shall define a notion for it.

  Definition 3.1. *Using Python's array slicing notation, we let*

$$c^{i_0:i_1, j_0:j_1, k_0:k_1}$$

where $i_0 < i_1$, $j_0 < j_1$, and $k_0 < k_1$ to be a grid of dimension $(i_1 - i_0) \times (j_1 - j_0) \times (k_1 - k_0)$ where

$$c_{ijk}^{i_0:i_1, j_0:j_1, k_0:k_1} = c_{i_0+i, j_0+j, k_0+k}$$

for all $0 \le i < i_1 - i_0$, $0 \le j < j_1 - j_0$, and $k < k_1 - k_0$.

- Notice that $c = c^{0:n,0:n,0:n}$.
- We will also discuss *subvolumes* of a volume.

  Definition 3.2. *Let $V : [x_0, x_1] \times [y_0, y_1] \times [z_0, z_1] \to \mathbb{R}$ be a volume function. We define*

  $$V^{[x_0', x_1'] \times [y_0', y_1'] \times [z_0', z_1']},$$

  where $x_0 \le x_0' \le x_1' \le x_1$, $y_0 \le y_0' \le y_1' \le y_1$, and $z_0 \le z_0' \le z_1' \le z_1$, to be another volume function from $[x_0, x_1] \times [y_0, y_1] \times [z_0, z_1]$ to $\mathbb{R}$ such that

  $$V^{[x_0', x_1'] \times [y_0', y_1'] \times [z_0', z_1']}(x, y, z) = \begin{cases} V(x, y, z), & \text{if } (x, y, z) \in [x_0', x_1'] \times [y_0', y_1'] \times [z_0', z_1'] \\ 0, & \text{otherwise} \end{cases}$$

- We will also discuss functions that takes a grid and turns it into a volume.

  Definition 3.3. *An function $\mathbf{V}$ is called a* grid-to-volume operator *if it satisfies the following properties:*

  - *$\mathbf{V}$ takes as input a grid c. Its output is denoted by $\mathbf{V}(c)$ or just $\mathbf{V}c$.*
  - *$\mathbf{V}(c)$ is a functiom from $[0, 1] \times [0, 1] \times [0, 1]$ to $\mathbb{R}$.*
  - *$\mathbf{V}$ is a linear function in c.*
  - *$\mathbf{V}$ "stretches" c so that it fills the cube $[0, 1] \times [0, 1] \times [0, 1]$. That is, the point $(0, 0, 0)$ coincides with c_{000} and the point $(1, 1, 1)$ coincides with c_{nnn}.*
  - *The value of a subvolume of $\mathbf{V}$ that coincides with a subgrid of c depends only on that subgrid. More precisely, let c be a grid of size $n + 1$. Then, the values of*

    $$\mathbf{V}(c)^{[i_0/n, i_1/n] \times [j_0/n, j_1/n] \times [k_0/n, k_1/n]}$$

    depends only on

    $$c^{i_0:i_1+1, j_0:j_1+1, k_0:k_1+1}.$$

  In general, we denote such a function with a boldfaced name such as $\mathbf{V}$.

- An example of such an operator is the one that takes a grid c of size $n + 1$, subdivides the unit cube into $n \times n \times n$ smaller cubes, and let the value of the (i, j, k)-cube be c_{ijk}. We denote this operator by **const**.
- The main grid-to-volume operator we will work on is the trilinear interpolation, which we shall denote by **tri**. Note that **tri** satisfies all the properties above. Moreover, the last section gave the closed form formula for line integrals through $\mathbf{tri}(c)$ where c is a grid of size 2.
- A volume function can be scaled and translated. We are particularly interested in the following operator.

Definition 3.4. *For $i, j, k \in \{0, 1\}$, define the operator $\mathbf{th}_{ijk}$ (standing for "half and then translate") as follows. If $V : [0,1] \times [0,1] \times [0,1] \to \mathbb{R}$ is a volume function, then $\mathbf{th}_{ijk}V$ is another volume function from $[0,1] \times [0,1] \times [0,1]$ to $\mathbb{R}$ such that*

$$(\mathbf{th}_{ijk}V)(x,y,z) = \begin{cases} V(2x-i, 2y-j, 2z-k), & \text{if } (x,y,z) \in [\frac{i}{2}, \frac{i+1}{2}] \times [\frac{j}{2}, \frac{j+1}{2}] \times [\frac{k}{2}, \frac{k+1}{2}] \\ 0, & \text{otherwise} \end{cases}.$$

The idea of $\mathbf{th}_{ijk}$ is to scale the volume down by a factor of 2 and translate it so that the lower-left-front corner is at $(i/2, j/2, k/2)$.

- The following lemma gives a way to decompose a volume function constructed from a grid to the volume functions constructed from the grid's octants. We state it without proof.

Lemma 3.5. *If c is a grid of size $2m+1$, and let $\mathbf{V}$ be a grid-to-volume operator discussed above, then*

$$\begin{aligned}
\mathbf{V}(c)^{[0,0.5]\times[0,0.5]\times[0,0.5]} &= \mathbf{th}_{000}\mathbf{V}(c^{0:m+1,0:m+1,0:m+1}), \\
\mathbf{V}(c)^{[0.5,1]\times[0,0.5]\times[0,0.5]} &= \mathbf{th}_{100}\mathbf{V}(c^{m:2m+1,0:m+1,0:m+1}), \\
\mathbf{V}(c)^{[0,0.5]\times[0.5,1]\times[0,0.5]} &= \mathbf{th}_{010}\mathbf{V}(c^{0:m+1,m:2m+1,0:m+1}), \\
\mathbf{V}(c)^{[0.5,1]\times[0.5,1]\times[0,0.5]} &= \mathbf{th}_{110}\mathbf{V}(c^{0:m+1,m:2m+1,0:m+1}), \\
\mathbf{V}(c)^{[0,0.5]\times[0,0.5]\times[0.5,1]} &= \mathbf{th}_{001}\mathbf{V}(c^{0:m+1,0:m+1,m:2m+1}), \\
\mathbf{V}(c)^{[0.5,1]\times[0,0.5]\times[0.5,1]} &= \mathbf{th}_{101}\mathbf{V}(c^{m:2m+1,0:m+1,m:2m+1}), \\
\mathbf{V}(c)^{[0,0.5]\times[0.5,1]\times[0.5,1]} &= \mathbf{th}_{011}\mathbf{V}(c^{0:m+1,m:2m+1,m:2m+1}), \text{ and} \\
\mathbf{V}(c)^{[0.5,1]\times[0.5,1]\times[0.5,1]} &= \mathbf{th}_{111}\mathbf{V}(c^{m:2m+1,m:2m+1,m:2m+1}).
\end{aligned}$$

3.2 The Framework

- Let c be a grid of size $2^k + 1$ for some $k \in \mathbb{Z}^+ \cup \{0\}$, and let $\mathbf{V}$ be a grid-to-volume operator. We are interested in constructing a data structure that can answer the following query:

  Given two points $\mathbf{p}_0$ and $\mathbf{p}_1$ in $[0,1] \times [0,1] \times [0,1]$, compute the line integral:

$$\|\mathbf{p}_1 - \mathbf{p}_0\| \int_0^1 (\mathbf{V}(c))(\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)t) \, \mathrm{d}t. \tag{4}$$

  Let us calls this a *volume line integral data structure* (VLIDS).

- We start by showing that a data that computes (4) can be obtained from ones that answers a slightly different but easier query:

  Given two points $\mathbf{p}_0$ and $\mathbf{p}_1$ *on the boundary* of $[0,1] \times [0,1] \times [0,1]$, compute the line integral:

$$\|\mathbf{p}_1 - \mathbf{p}_0\| \int_0^1 (\mathbf{V}(c))(\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)t) \, \mathrm{d}t. \tag{5}$$

  Let us call this data structure a *surface point line integral data structure* (SPLIDS).

- A SPLIDS represents a 4D function $s(\mathbf{p_0}, \mathbf{p_1})$ that maps two points on the boundary of the unit cube to the line integral of the segment between them. We shall refer to this kind of functions as the *surface point line integral functions* (SPLIF).

- **Lemma 3.6.** *Let all the input sizes in this lemma be powers of 2. If there is a SPLIDS that takes $f(n)$ time to answer a query and takes $g(n)$ total space for a grid of size $n+1$, then there is a VLIDS that takes*

$$O(f(1) + f(2) + \cdots + f(2^r) + \cdots + f(n))$$

time to answer a query and

$$O(n^3 g(1) + n^3 g(2)/8 + \cdots + n^3 g(2^r)/2^{3r} + \cdots + g(n))$$

total space.

Proof. Note that a grid of size $n+1$ partitions the unit cube to n^3 cells. We build an octree on these n^3 cells such that each leaf corresponds to a cell. For each node in the octree, we build a SPLIDS for the volume corresponding to the node. This yields a data structure with

$$O(n^3 g(1) + n^3 g(2)/8 + \cdots + n^3 g(2^r)/2^{3r} + \cdots + g(n))$$

space.

Given two points $\mathbf{p}_0$ and $\mathbf{p}_1$ inside the unit cube, we answer the query as follows. From the data structure, we can find a sequence of non-intersecting volumes with the following properties:

- The volumes in the sequence are non-intersecting.
- The volumes together covers the line segment from $\mathbf{p}_0$ to $\mathbf{p}_1$.
- Each volume corresponds to a node in the octree.
- There are at most 4 volumes in the sequence belonging to each level of the octree.

The procedure of finding such a sequence of volumes is pretty much like tracing a ray from $\mathbf{p}_0$ to $\mathbf{p}_1$ in an octree. We can then use the SPLIDS for each volume to compute the line integral of the part of the segment that goes through the volume, and then add all the result together. This process thus takes $O(f(1) + f(2) + \cdots + f(2^r) + \cdots + f(n))$ as desired. □

- We still need to determine the time needed to create all the SPLIDSs according to the data structure outlined in the last Lemma. We would prefer not to create the SPLID for an octree node from the underlying grid data. Instead, we would like to build it from the SPLIDSs of the node's children. If we can do so, we can characterize the running time of the preprocessing phase as follows:

  Lemma 3.7. *If the SPLIDS for an octree node corresponding to a subvolume of size $n+1$ can be built from the 8 SPLIDS of the node's children in time $h(n)$, then the whole data structure in Lemma 3.6 can be built in*

$$O(n^3 + n^3 h(2)/8 + \cdots + n^3 h(2^r)/2^{3r} + \cdots + h(n)),$$

assuming that SPLIDS for a grid of size 2 (the leaf node) can be built in constant time.

Proof. Just count the number of SPLIDS to be built in each level. □

- We want an implementation of SPLIDSs allow us to accomplish the following tasks:

  (i) Given a SPLIDS of a volume function V, compute the SPLIDS of $\mathbf{th}_{ijk}V$ for any $i, j, k \in \{0, 1\}$.
  (ii) Given SPLIDSs for volume functions V_1 and V_2, compute the SPLIDS of $V_1 + V_2$.

  If we can do so, then we create create the SPLIDS of a node from the SPLIDSs of its children. This is the result of Lemma 3.5 and the fact that the SPLIF is linear in its underlying volume.

- We aim to make $f(n) = O(1)$ so that the lookup time is $O(\log n)$, and $g(n) = h(n) = O(n^2)$ so that the space used and the precessing time is

  $O(n^3 \cdot 1 + n^3 \cdot 4/8 + n^3 \cdot 16/64 + \cdots + n^3 \cdot 2^{2r}/2^{3r} + \cdots + n^2) = O(n^3 + n^3/2 + \ldots + n^3/2^r + \ldots + n^2) = O(n^3)$

  so that the precomputed data do not take more space than the original data.

4 Possible SPLIDS Implementations

4.1 Lookup Tables

- We can implement a SPLID as a table of equally spaced samples of the SPLIF it represents. Reconstruction involves doing a quadrilinear interpolation of the stored samples. Hence, a query can be answered in constant time.
- The implementation would be to define a resolution parameter m. Then, we put an $m \times m$ grid of sampling points on each face of the cube. This results result in $6m^2$ points, and we can make a lookup table of size $6m^2 \times 6m^2 = 36m^4$.
- However, the space requirement can be reduced by exploiting basic properties of the SPLIF, which we shall denote with s.

  First, if $\mathbf{p_0}$ and $\mathbf{p}_1$ lie on the same face of the cube, then $s(\mathbf{p}_0, \mathbf{p}_1) = 0$. Thus, there is no need to store the sample whose points lie on the same face.

  Second, the SPLIF is a symmetric function. That is, $s(\mathbf{p}_0, \mathbf{p}_1) = s(\mathbf{p}_1, \mathbf{p}_0)$. Hence, we can totally order the faces and store only those samples that go from a "lower" face to "higher" faces.

  Exploting the above properties, we store the sample in $\binom{6}{2} = 15$ tables of size $m^2 \times m^2$. The space requirement is then $15m^4$.
- To facilitate interpolation, we recommend positioning the grid of sample points so that the grid's edge coincide with the cube's face edge. In this way, we have that for any other points on the face, we can always find 4 sampling points forming a rectangle around that point.
- In conclusion, the loopup table takes $O(1)$ to answer a query and $O(m^4)$ space. The space requirement makes it impractical. That is, for a volume grid of size n, it would be reasonable to set $m = \Theta(n)$, but $O(n^4)$ space for the SPLIDS is not reasonable at all.

4.2 Factored Matrices

- In the last section, a SPLIDS is represented by 15 lookup tables of size $m^2 \times m^2$. We can think of each lookup table as a matrix A and approximate it with a rank-r approximation $A \approx UV^T$ where $U, V \in \mathbb{R}^{m^2 \times p}$ for some $r << m^2$.
- Using this representation, a lookup takes $O(r)$ and the space requirement is $O(m^2 r)$, which is a huge saving from $O(m^4)$.
- One drawback is that this representation is that the reconstruction would not be accurate if the underlying SPLIF is not a low rank signal. However, we believe that, if the data in the volume grid is low in frequency, the SPLIF should be represented well by a low rank approximation.
- In order to use this SPLIDS implementation in the framework outlined in Section 3.2, we must be able to

  (a) Construct a SPLIDS for a grid of size 2.

  (b) Combine 8 SPLIDSs for octants of a volume into the SPLIDS for the volume itself (which involves task (i) and (ii) in Section 3.2).
- Requirement (a) is easy to satisfy. We may precompute the full lookup tables for V_{000}, V_{001}, ..., V_{111} for some small resolution parameter m, and then compute a linear combination of them to get the full lookup table for any grid of size 2. We can then factor the resulting matrix.

- Requirement (b) is harder to satisfy as factored matrices cannot be directly added together. Moreover, we don't know how to compute the factored matrix of $\mathbf{th}_{ijk}V$ from the factored matrix of V without expanding the matrix and multiplying it with the matrix associated with $\mathbf{th}_{ijk}$, which is much bigger (its size is $O(m^8)$).
- Rather, we propose satisfying Requirement (b) by a data-driven approach. We view computing the SPLIDS as a matrix approximation problem. In literature, there are already a number of algorithms that can approximate a matrix by sampling a few of its rows and columns. [3, 1].
- More specifically, if we have 8 SPLIDS of children of an octree node, then we can evaluate the SPLIF of the parent by breaking the line segment connecting the end points into parts, each of which intersects only one child. We can then use the SPLIDS of the children to evalute the line integral for each segment and add the results up to get the parent's SPLIF value. This evaluation can be done in $O(r)$ time. Hence, it is possible to sample a row or a column of the parent's SPLIF matrix in $O(m^2r)$ time.
- The Nystrom method [3], for example, samples r rows and r columns of the matrix to get a rank-r approximation. In our settings, these sampling can be done in $O(m^2r^2)$ time. Additional $O(m^2r^2+r^2)$ time is required to produced the low rank approximation. (This includes multiplying an $r \times m^2$ matrix to a $m^2 \times r$ matrix, and finding the pseudo-inverse of an $r \times r$ matrix.)
- Therefore, if we set the resolution parameter m to be $O(n)$, then we have $f(n) = O(r)$, $g(n) = O(n^2r)$, and $h(n) = O(n^2r^2 + r^3)$. Since r is a constant, we have an implement of SPLIDS that satisfies all the goals we were aiming for in Section 3.2.
- However, one worrying aspect of this implementation is that we do not know how to assess the accuracy of the implementation.

4.3 Spherical Harmonic Reflection Map

- The spherical harmonic reflection map (SHRM) was invented by Ramamoorthi and Hanrahan to represent isotropic BRDFs [2]. An SHRM is a cube map whose entries are spherical harmonoics. In other words, each sample point on the surface of the cube stores a 2D function. Since the SHRM is capable of representing any 4D function, it can be used to represent SPLIF as well.
- We use the SHRM to store the SPLIF as follows. Let the cube map have resolution $m \times m$ where m is the resolution parameter. For each sample point $\mathbf{p}$ on a cube's face, we store the spherical function

$$Y_{\mathbf{p}}(\omega) = s(\mathbf{p}, l(\mathbf{p}, \omega))$$

  where $l(\mathbf{p}, \omega)$ is the first point that the ray $\mathbf{p} + t\omega$ intersects the unit cube.
- Evaluating $s(\mathbf{p}_0, \mathbf{p}_1)$ involves finding 4 sample points $\mathbf{p}_{BL}$, $\mathbf{p}_{BR}$, $\mathbf{p}_{TL}$, $\mathbf{p}_{TR}$ forming a rectangle around $\mathbf{p}_0$, and then doing a bilinear interpolation between

$$\begin{aligned}
&Y_{\mathbf{p}_{BL}}(\mathbf{p}_{BL}, (\mathbf{p}_1 - \mathbf{p}_{BL})/\|\mathbf{p}_1 - \mathbf{p}_{BL}\|),\\
&Y_{\mathbf{p}_{BR}}(\mathbf{p}_{BR}, (\mathbf{p}_1 - \mathbf{p}_{BR})/\|\mathbf{p}_1 - \mathbf{p}_{BR}\|),\\
&Y_{\mathbf{p}_{TL}}(\mathbf{p}_{TL}, (\mathbf{p}_1 - \mathbf{p}_{TL})/\|\mathbf{p}_1 - \mathbf{p}_{TL}\|), \text{ and}\\
&Y_{\mathbf{p}_{TR}}(\mathbf{p}_{TR}, (\mathbf{p}_1 - \mathbf{p}_{TR})/\|\mathbf{p}_1 - \mathbf{p}_{TR}\|).
\end{aligned}$$

- If we use a spherical harmonic representation of order q, then the representation uses $6m^2q^2$ floating points, and a query can be answered in $O(q^2)$ time.
- One nice thing about the SHRM is that, if have an SHRM that represents the SPLIF of V_1 and another that represents the SPLIF of V_2, we can compute the SHRM that represents the SPLIF of $V_1 + V_2$ by just adding the two SHRMs together. This makes conbining children's SPLIDS when creating parent's SPLIDS very easy.

- Moreover, suppose we have an SHRM representation of the SPLIF of V, and this representation is at resolution m'. Then, there is a linear operator that transforms it to the SHRM representation of the SPLIF of $\mathbf{th}_{ijk}V$ at resolution m. This linear operator is a matrix of dimension $m^2q \times m'^2q$. Let us denote this matrix by $\mathbf{TH}_{ijk}^{m' \to m}$. We can precompute them for all i, j, k and various m' and m.

- Naively storing $\mathbf{TH}_{ijk}^{m' \to m}$ uses $O(m^2m'^2q^4)$ space, which is not scalable if m and m' are large. To alleviate this problem by storing and using its rank-r approximation. This reduces the space requirement to $O((m^2+m'^2)q^2r)$. It also reduces the time requirement of transforming the SHRM to $O((m^2+m'^2)q^2r)$ as well.

- All in all, setting $m = O(n)$, we have that $f(n) = O(q^2)$, $g(n) = O(n^2q^2)$, and $h(n) = O(n^2q^2r)$. We thus have another representation that satisfies all the requirements of the framework in Section 3.2.

- However, this SPLIDS implementation can only represent low frequency SPLIF, and we don't know how to quantify the error it introduces.

References

[1] Matthew O'Toole and Kiriakos N. Kutulakos. Optical computing for fast light transport analysis. In *ACM SIGGRAPH Asia 2010 papers*, SIGGRAPH ASIA '10, pages 164:1–164:12, New York, NY, USA, 2010. ACM.

[2] Ravi Ramamoorthi and Pat Hanrahan. Frequency space environment map rendering. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 517–526, New York, NY, USA, 2002. ACM.

[3] Jiaping Wang, Yue Dong, Xin Tong, Zhouchen Lin, and Baining Guo. Kernel nystrom method for light transport. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 29:1–29:10, New York, NY, USA, 2009. ACM.