System for Generating Radiometric Sub-Pixel Targets

Jack Pohlmann

April 2019

Abstract

This prospectus introduces the framework of a system designed to quickly generate remotely-sensed hyperspectral pixels with sub-pixel targets in the infrared. The bulk of this document focuses on the derivation of what is called the *second-order rendering equation*. This equation describes how a generalized three-dimensional target can be integrated into a background pixel utilizing second-order reflections.

Introduction

This intent of this document is to describe the methodology for a system designed to generate individual hyperspectral signatures. The system focuses on generating sub-pixel targets at off-nadir viewing angles[1] while striving to balance high physical fidelity with computation speed.

The document begins with a derivation of the specialized physics used in internal calculations. Although the equations are broadly applicable, they are derived with the intent of being used by this system. Following the physical discussion is an abstract description of the internal data interfaces. This is a natural continuation after the physics because it ties physical quantities to internal exchanges. Finally, the document discusses the high-level implementation of dependencies, plugins, and default tuning parameters.

This document does not discuss low-level implementations or explicit code.

1 Specialized Physics

The focus of this section is to walk through the derivation of the computational physics used in the generation system. Unless specified otherwise, all radiometric quantities (eg. radiance, reflectance, etc.) are assumed to be spectrally dependent. In order to streamline the equations, the explicit spectral variables are omitted.

1.1 First-order radiometric equation

In this application, the first-order (ie. pure, flat pixel) rendering equation[2] is given as:

$$L_1(\theta_o, \phi_o) = L_e(\theta_o, \phi_o) + \int_{2\pi} \int_{\pi/2} \rho(\theta_o, \phi_o, \theta_i, \phi_i)\ L_i(\theta_i, \phi_i) \sin(\theta_i) \cos(\theta_i) d\theta_i d\phi_i$$

where L_e is the self-emitted radiance, ρ is the hemispherical reflectance,[3] and L_i is the incoming radiance. θ_o, ϕ_o and θ_i, ϕ_i are the observed and incoming zenith and azimuth angles, respectively. It is necessary to note that in the first-order equation no term is specified as a vector quantitiy since all are assumed to have a nadir normal.

[1] As a consequence of the necessary computation, the system is equally adept at generating signatures for any viewing angle, and can generate an entire hemispherical signature.

[2] The equation is called the rendering equation because it is the typical equation used when rendering scenes in graphical applications. [Course Notes; Marciniak 2018]

[3] A common example of a hemispherical reflectance is the Bidirectional Reflectance Distribution Function (BRDF). The BRDF is not explicitly used as the hemispherical reflectance because it can be substituted with more simplified hemispherical distribution functions.

The rendering equation is discretized in the typical manner – exchanging the integral for a discrete sum:

$$L_1(\theta_o, \phi_o) = L_e(\theta_o, \phi_o) + \sum_{\phi_i}^{2\pi} \sum_{\theta_i}^{\pi/2} \rho(\theta_o, \phi_o, \theta_i, \phi_i)\ L_i(\theta_i, \phi_i)\ \sin(\theta_i)\cos(\theta_i)\Delta\theta\Delta\phi\ . \tag{1.1}$$

The most important change is $d\theta_i d\phi_i \rightarrow \Delta\theta\Delta\phi$; the differential is no longer tied to the incoming view angles. Rather, they determine the *resolution* of the associated angles. The angular resolutions are discussed further in the section on system tuning parameters.

1.2 Multi-facet targets

The first-order rendering equation assumes a pure, flat pixel. Adapting this to a flat mixed pixel is simple using linear mixing:

$$\begin{aligned} L_e \longrightarrow L_e &= \alpha\ L_{targ} + (1-\alpha)\ L_{bg} \\ \rho \longrightarrow \rho &= \alpha\ \rho_{targ} + (1-\alpha)\ \rho_{bg} \end{aligned}$$

where the constant α is the pixel coverage ratio, or mixing ratio, with respect to nadir. The usage of linear mixing removes any dependence on sub-pixel relative coordinates. Using the above transformations with Eq. 1.1, the first-order sub-pixel radiance becomes:

$$L_{1,o} = \alpha\ L_{1,o,targ} + (1-\alpha)\ L_{1,o,bg}\ .$$

In order to construct a higher fidelity model the next logical inclusion is a three-dimensional target. In real-world scenarios, most sub-pixel targets will be explicitly macroscopic. A simple way to describe an opaque macroscopic target is to write it as a surface with multiple faces or facets. The total target radiance is then just the sum of the radiance of each facet:

$$L_{mf,e} = \sum_{facets} \beta_{facet}(\theta_o, \phi_o)\ L_{1,facet}(\theta_o, \phi_o)$$

where $\beta_{facet}(\theta_0, \phi_o)$ is the the sub-coverage ratio of the facet. Substituting this into Eq. 1.1 for a target with N facets the total radiance with respect to nadir is found to be:

$$L_{mf}(\theta_o, \phi_o) = \sum_{n=1}^{N} \beta_n(\theta_o, \phi_o)\ L_{1,targ,n}(\theta_o, \phi_o) + (1-\alpha(\theta_o, \phi_o))\ L_{1,bg}(\theta_o, \phi_o)\ . \tag{1.2}$$

Here, $\alpha(\theta_o, \phi_o) = \sum_{n=1}^{N} \beta_n(\theta_0, \phi_o)$ is the *projected* coverage ratio, now dependent on viewing angle, and β_n is the sub-coverage ratio of the nth facet. A subscript n indicates that the quantity is linked to the nth facet which further implies that it may not have a normal parallel to nadir. Furthermore, L_1 indicates the first-order rendered radiance as given by Eq. 1.1. In the case of the facet radiance, the first-order rendering equation of the nth facet becomes:

$$L_{1,targ,n}(\vec{\theta_o}, \vec{\phi_o}) = L_{e,targ}(\vec{\theta_o}, \vec{\phi_o}) + \sum_{\theta_i=0}^{2\pi} \sum_{\phi_i=0}^{\pi/2} \rho(\theta_o, \phi_o, \theta_i, \phi_i)\ L_{i,n}(\vec{\theta_i}, \vec{\phi_i})\ \sin(\theta_i)\cos(\theta_i)\Delta\theta\Delta\phi. \tag{1.3}$$

While nearly identical to Eq. 1.1, the vector notation again indicates a normal different from nadir,[4] and the lack of vector notation indicates that the normal is irrelevant in calculation. With this in mind, it should be clear that the incoming *facet* radiance $L_{i,n}(\vec{\theta_i}, \vec{\phi_i}) \neq L_i(\theta_i, \phi_i)$, the incoming *nadir* radiance.

[4] In essence, if the hemisphere normal to the facet surface intersects the surface plane then some incoming radiance will come from the first-order rendered background radiance.

1.3 Second-order radiometry

Combining Eq.s 1.2 and 1.3 yields the initial second-order radiance equation:

$$L_2(\theta_o, \phi_o, ...) =$$
$$\sum_{n=1}^{N} \beta_n(\theta_o, \phi_o) \Bigg[L_{e,targ}(\theta_{o,n}, \phi_{o,n}) + \sum_{\theta_i=0}^{2\pi} \sum_{\phi_i=0}^{\pi/2} \rho(\theta_{o,n}, \phi_{o,n}, \theta_i, \phi_i)\ L_{i,n}(\theta_{i,n}, \phi_{i,n})\ \sin(\theta_i)\cos(\theta_i)\Delta\theta\Delta\phi \Bigg] +$$
$$(1 - \alpha(\theta_o, \phi_o)) \Bigg[L_{e,bg}(\theta_o, \phi_o) + \sum_{\theta_i=0}^{2\pi} \sum_{\phi_i=0}^{\pi/2} \rho(\theta_o, \phi_o, \theta_i, \phi_i)\ L_i(\theta_i, \phi_i)\ \sin(\theta_i)\cos(\theta_i)\Delta\theta\Delta\phi \Bigg] . \quad (1.4)$$

The vector notation in Eq.s 1.2 and 1.3 has been substituted for facet subscription. Unfortunately, this representation is not complete since L_2 should only be a function of the observed or outgoing angles. To complete this equation, a transformation of the facet normal coordinates $\theta_{x,n}, \phi_{x,n}$ must first be performed, where subscript x denotes either observed or incoming angles.

1.4 Geometric transformations

The transformation from *nadir-normal* coordinates to purely *nadir* coordinates is straightforward. This transformation is necessary to align all computations which drastically reduces complexity. For a facet normal $\vec{n}$ and the nadir normal $\vec{n}_0$, the coordinate change is:

$$(\theta_x, \phi_x) \longrightarrow (\theta_{x,n}, \phi_{x,n}) - (\vec{n} - \vec{n}_0) .$$

The vector difference $(\vec{n} - \vec{n}_0)$ can be written as the *angle shifts* $(\delta\theta_n, \delta\phi_n)$, leading to a more generalized coordinate change:

$$(\theta_{x,n}, \phi_{x,n}) \longrightarrow (\theta_x + \delta\theta_n, \phi_x + \delta\phi_n) . \quad (1.5)$$

In order to geometrically interpret the angle shifts it is first necessary to explicitly define the orientation of the different L_2 radiance dependencies L_i and $L_{1,bg}$. The incoming radiance L_i is computed as an irradiance would be in that the dependent angles θ_i and ϕ_i point in the direction of *incoming* rays; in other words, the radiance points *inward* from the surface of a hemisphere. Conversely, the background radiance $L_{1,bg}$ is computed as an exitance would be such that θ_o and ϕ_o point along *outgoing* rays; or, that the radiance points *outward* from the surface of a hemisphere. This is sensible since $L_{1,bg}$ is simply the L_1 radiance for a pure pixel calculated using the reflectance distribution ρ integrated along a normal hemisphere, in this case L_i. These distinctions play an essential role in computing $L_{i,n}$, the *facet normal radiances*.

Consider the case $\delta\theta_n > 0$ which implies that there exists some $\theta_x \in [0, \pi/2]$ such that $\theta_{x,n} = \theta_x + \delta\theta_n > \pi/2$. Simply stated, a nonzero zenith shift pushes the zenith angle range past the horizontal plane. The outgoing facet zenith angle $\theta_{o,n}$ is constrained such that $\theta_{o,n} \leq \pi/2 - \delta\theta_n$ is *always* true – the system is unconcerned with underground sensors.

When computing the second-order reflections of the background (ie. backscattering), however, looking towards the ground is required. In the normal zenith angle range $[0, \pi/2]$, both $\theta_{x,n}$ and $\phi_{x,n}$ point towards the *inner* side of the *outer* hemisphere defined by L_i. This yields a direct correspondence to the inward facing rays required by $L_i(\theta_i, \phi_i)$. In the case where the zenith angle moves past horizontal, $\theta_{i,n} > \pi/2$ with respect to nadir, the view angles point towards the *outer* side of the *inner* hemisphere defined by $L_{1,bg}$, the first-order radiance of the background. In order to flip this hemisphere to be an *inner-outer* hemisphere like L_i the coordinates need to be changed as such:

$$\theta_{i,n} \longrightarrow \pi - \theta_{i,n}$$
$$\phi_{i,n} \longrightarrow \pi - \phi_{i,n}$$

which leads to the complete transformation of the facet coordinates written as functions of the nadir viewing

angles:

$$(\theta_{o,n}, \phi_{o,n}) = (\theta_o + \delta\theta_n, \phi_o + \delta\phi_n);\ \theta_o \in [0, \pi/2 - \delta\theta_n]$$
$$(\theta_{o,n}, \phi_{o,n}) = \begin{cases} (\theta_i + \delta\theta_n, \phi_i + \delta\phi_n) & \mid\ \theta_i \in [0, \pi/2 - \delta\theta_n) \\ (\pi - \theta_i - \delta\theta_n, \pi - \phi_i - \delta\phi_n) & \mid\ \theta_i \geq \pi/2 - \delta\theta_n \end{cases} \tag{1.6}$$

This definition of the view angles allows the facet normal radiance to be transformed into the *facet nadir radiance*:

$$L_{i,n}(\theta_i, \phi_i) = \begin{cases} L_i(\theta_i + \delta\theta_n, \phi_i + \delta\phi_n) & \mid\ \theta_i \in [0, \pi/2 - \delta\theta_n) \\ L_{1,bg}(\pi - \theta_i - \delta\theta_n, \pi - \phi_i - \delta\phi_n) & \mid\ \theta_i \geq \pi/2 - \delta\theta_n\ . \end{cases} \tag{1.7}$$

Finally, in order to compensate for the truncated observeation angles given in Eq. (6), the facet projected areas $\beta_n(\theta_o, \phi_o) = 0$ for $\theta_o \notin [0, \pi/2 - \delta\theta_n]$. This truncation is equivalent to ignoring backscatter from the target onto the background and allows the restriction on θ_o to be lifted. Applying Eq.s 1.6 and 1.7 to Eq. 1.4 yields the complete second-order rendering equation:

$$L_2(\theta_o, \phi_o) = \sum_{n=1}^{N} \beta_n(\theta_o, \phi_o) \Big[L_{e,targ}(\theta_o + \delta\theta_n, \phi_o + \delta\phi_n) + \sum_{\theta_i=0}^{2\pi} \sum_{\phi_i=0}^{\pi/2} \rho(\theta_o + \delta\theta_n, \phi_o + \delta\phi_n, \theta_i, \phi_i)\ L_{i,n}(\theta_i, \phi_i)\ \sin(\theta_i)\cos(\theta_i)\Delta\theta\Delta\phi \Big] +$$
$$\big[1 - \alpha(\theta_o, \phi_o)\big] \Big[L_{e,bg}(\theta_o, \phi_o) + \sum_{\theta_i=0}^{2\pi} \sum_{\phi_i=0}^{\pi/2} \rho(\theta_o, \phi_o, \theta_i, \phi_i)\ L_i(\theta_i, \phi_i)\ \sin(\theta_i)\cos(\theta_i)\Delta\theta\Delta\phi \Big]\ . \tag{1.8}$$

In the case of a flat (single-faceted) target, $\beta_{N=1} = \alpha,\ \delta\theta_{N=1} = \delta\phi_{N=1} = 0,\ L_{i,N=1} = L_i$, and it is therefore simple to show that L_2 collapses to the linearly mixed case of L_1.

2 Data Interfacing

This section seeks to explain the manner in which data is exchanged within sections of the generation system. The need for a defined interface, or common language, is twofold:

First, the goal for modularity in this system[5] demands that modules or plugins can communicate to other systems in the same manner as their counterparts. A simple way to describe this is to look at the hemispherical reflectance distribution function, or ρ throughout Section 1. ρ can represent a multitude of distributions from a simple Lambertian constant to a complex Sandford-Robertson BRDF. Regardless of its complexity, ρ always has units of sr^{-1}. Therefore, a function utilizing ρ such as Eq. 1.4 is agnostic to complexity – it only cares that it receives the correct data type.

Second, it is considered best practice in any large programming application to maintain internal interfaces because it locally constrains bugs, tests, and development. For example, if a computational unit is isolated from another, then changing or updating the unit should have no impact on the other unit. This facilitates the growth of the system since complexity can be added or removed at will without much effort.

This section begins with physical interfacing much like the hemispherical reflectance distribution example above. Following that is a description of geometrical interfacing focusing on target abstraction. The section concludes with a brief overview of computational interfacing.

2.1 Physical interfacing

This section quickly runs through the physical quantites in Eq. 1.4. The recipe for each componenent is to define the most generalized functions for each quantity. Generalizing the functions ensures that even the most complex version of each quantity can be used by the system without trouble. In situations where the

[5]Modularity improves the capability of the system to grow upward, outward, or to integrate with other systems. Furthermore, it lets users try out different modules or plugins using pregenerated data rather than running the entire system each time.

simplest version of a quantity is used, extraneous variables can simply be ignored. As mentioned in the Section 1 introduction, every physical quantity is spectrally dependent so this continues to be ignored.

As a preemptive conclusion, generalizing the physical quantities is incredibly straightforward and simplifies the internal computation drastically. It encourages the internal logic to be the responsibility of each computational component.

Radiance, L

Radiances used in the system are explicitly incoming or outgoing as previously mentioned. However, this distinction can easily be ignored and kept track of solely through context. Therefore, the generalized form (ie. interface) of the radiance is:

$$L = L(\theta, \phi) \ .$$

Note that the method for producing radiances or any other physical quantity may differ. This is best exemplified in Section 3.

Hemispherical reflectance distribution, ρ

The hemispherical reflectance distribution is more complex than the radiances because it relates incoming angles to outgoing angles. This is easily generalized as

$$\rho = \rho(\theta_o, \phi_o, \theta_i, \phi_i)$$

where subscript o and i indicate incoming and outgoing, respectively.

Projected areas, α and β

The projected areas are again simple to generalize. In fact, distinguishing between α and β is only useful analytically. Internally, the generalization is

$$\beta = \beta(\theta, \phi)$$

where the projected coverage ratio α is just a special case of the facet coverage ratio β above. α can be computed as $\alpha = \sum_{facets} \beta_{facet}$ or given, and the method for requesting computation or supplying the data is overviewed at a high level in Section 2.3.

2.2 Geometric interfacing

The focus of this section is to use an example to outline the geometric interface. Specifically, the geometric interface refers to the facet notation of the target scene. As a consequence, this section crosses over with Section 3.2, the section on modules and plugins, and therefore serves a dual example.

The setup for this scenario is as follows: a simple three-dimensional target and flat, pure background.

Target layout

[6] The target in this scenario is represented by the simplest three-dimensional target case: a cube. In this scenario, a cube target has $n = 5$ facets; one for each face minus the face in contact with the background. The facets can be parameterized as follows:

$$\begin{aligned} n_n &= (\delta\theta_n, \delta\theta_n) \\ n_0 &= (0, 0) \\ n_1 &= (\pi/2, 0) \\ n_2 &= (\pi/2, \pi/2) \\ n_3 &= (\pi/2, \pi) \\ n_4 &= (\pi/2, 3\pi/2) \ . \end{aligned}$$

[6] Still need diagrams.

Each facet carries with it a projected coverage ratio β, a self-emitted radiance L_e, a hemispherical reflectance distribution ρ, and an incoming radiance L_i. Of these quantities, L_e and ρ are identical except for the offset given by f_n. The other quantities, β and L_i, can be computed easily with

$$\beta_n = l_n \times w_n \times \cos(\theta_{o,n})$$

where l_n and w_n are the nth facet length and width and $\theta_{o,n}$ is the facet viewing angle as given by Eq. 1.6. Likewise, the incoming radiance L_i is determined by Eq. 1.7.

Computation

Equipped with L_e, ρ, and a list containing each facet's β and L_i, the entire target is generalized and can be supplied to the component that uses them for computation. This can be extended to any macroscopic target, the only real difference being keeping track of the area for β, where l_n and w_n are replaced by some arbitrary facet area A_n.

With a background surface radiance and incoming sky radiance already given in the form specified by Section 2.1, or as $L_{bg} = L_{bg}(\theta, \phi)$ and $L_i = L_i(\theta, \phi)$, a rendering equation such as Eq. 1.4 can be computed directly and the system returns and terminates.

2.3 Computational interfacing

This section focuses on the manner in which data is exchanged between computational components. This differs from the previous interfacing methods in the sense that all data, including instructions and parameters, must be kept track of throughout the system. An expliciit ruleset for labeling data is important when trying to design and implement external plugins. As a result of the subject-matter, as above, this section has a lot of overlap with the follwoing sections describing the module/plugin structure of the system.

Without diving too deeply into the programming-specific mechanisms,[7] data is referenced between components by grouping data into a hierarchical data structure, or `hdstruct`. Along with the `hdstruct` is an `instruct`, or instruction structure,[8] grouped by module (ie. the mandatory computational components) that indicate the keys in the `hdstruct`.

In the previous section's example, the data would be grouped into the `hdstruct` under a tag along the lines of `'cube'`. This key would be indicated in the `instruct` under a key such as `'target'`. Here, a module that looks for information under with the key `'target'` would know to generate the data with a plugin designated by `'cube'`. That plugin would then use the data grouped in the `hdstruct` under `'cube'` to generate the data and return it to the `'target'` module which returns newly generated data.

While this method seems complicated – why ask `'target'` to call `'cube'` when the parent function could just do it – in the case where multiple plugins act on the target generation it is more intuitive to group those plugins with the `'target'` keyword. Then, it is completely clear which componenent utilizes which plugin.[9]

3 Functionality

This section focuses on the functional mechanisms on top of which the system is implemented. This includes examples of external dependencies, implementations of mandatory modules or supplemental plugins[10], and basic tuning parameters. The majority of these topics have been discussed previously in some capacity.

[7] Python is the main language in this application, so the specific mechanisms include objects such as dictonaries, lists, tuples, and NumPy arrays.

[8] Clever!

[9] Furthermore, if an external system wishes to query or inject data at different computation stages, it can be built into a plugin that can be directly called by the relevant components.

[10] The distinction between modules and plugins is as stated here: modules are required for the system to function. Although they can be exchanged, the lack of a module is fatal to the system's function. A plugin, on the other hand, adds extra functionality to the system. These can range from adding complexity or detouring the data-flow to external applications.

3.1 External dependencies

As the section title suggests, external dependencies are plugins that supply something, be it data or some sort of specialized computation, that is necessary for the

3.2 Modules and plugins

3.3 Tuning parameters