\( P_N \)-Method for Multiple Scattering in Participating Media

David Koerner\(^1\) Jamie Portsmouth\(^2\) Wenzel Jakob\(^3\)

\(^1\)University of Stuttgart \(^2\)Solid Angle \(^3\)École polytechnique fédérale de Lausanne (EPFL)

Abstract

Rendering highly scattering participating media using brute force path tracing is a challenge. The diffusion approximation reduces the problem to solving a simple linear partial differential equation. Flux-limited diffusion introduces nonlinearities to alleviate the approximation error but introduces several ad-hoc assumptions. Both methods are based on the spherical harmonics expansion of the radiance field, that is truncated after the first order. In this paper, we investigate the open question of whether higher orders provide a viable alternative to these two approaches. Increasing the order introduces a set of increasingly complex coupled partial differential equations, whose growing number and complexity make them very difficult to work with. We use a computer algebra framework for representing and manipulating the underlying mathematical equations and use it to derive the time-independent real-valued \( P_N \)-equations for arbitrary orders. We further present a staggered-grid \( P_N \)-solver and generate its stencil code directly from the expression tree of the \( P_N \)-equations. Finally, we discuss how our method compares to prior work for various standard problems. We will release our computer algebra system, solver, and data as open source to ensure reproducibility of all of our results.

1. Introduction

Simulating light transport in participating media remains a challenging problem for image synthesis in computer graphics. Due to their ability to produce unbiased results and conceptual simplicity, Monte Carlo based techniques have become the standard approach. The main downside of these methods are their computational demands when rendering media with strong scattering or anisotropy.

Deterministic methods have enjoyed less popularity, because they suffer from discretization artifacts, produce biased results, cannot be coupled easily with surface rendering problems and are trickier to implement. However, their appeal lies in the fact that they produce a global solution across the whole domain and have better performance for certain problems.

The work on path-guiding techniques from recent years has shown how approximate representations of the steady-state transport in a scene can be used to accelerate Monte Carlo integration techniques, such as path tracing. Instead of generating these approximate representations using Monte Carlo methods, deterministic methods may offer a viable alternative. Hybrid methods could combine the performance benefits of deterministic methods with accurate and unbiased Monte Carlo results.
Deterministic methods also lend themselves to applications where fast approximate solutions are preferable over correct, but slowly converging results. For this reason, we suggest it is important for volume-rendering researchers to study deterministic methods and have a solid understanding of their characteristics and performance traits for typical rendering problems.

The \( P_N \)-method is a deterministic method which is popular in fields like medical imaging and nuclear sciences, but has not found use in computer graphics thus far. The purpose and main contribution of our paper is to gain a solid understanding of its foundations and present a method for using it in the context of rendering. In particular, we present these theoretical and practical contributions:

- We derive and present the time-independent real-valued \( P_N \)-equations and write them down in a very concise and compact form which we have not found anywhere else in the literature.
- We introduce a staggered-grid solver, for which we generate stencil code automatically from a computer algebra representation of the \( P_N \)-equations. This allows us to deal with the increasingly complex equations which the \( P_N \)-method produces for higher order. It further allows our solver to be used for any (potentially coupled) partial differential equations, which result in a system of linear equations after discretization.
- Finally, we compare the \( P_N \)-method for higher orders against flux-limited diffusion and ground truth Monte Carlo integration.

In the next section, we will discuss related work and its relation to our contribution. In Section 3 we revisit the deterministic approach to light transport simulation in participating media and outline the discretization using spherical harmonics. In Section 4 we introduce our computer algebra representation, which we required to derive the real-valued \( P_N \)-equations. This representation is also a key component of our solver, which we present in Section 6. Section 7 discusses application of the solution in the context of rendering. We compare our \( P_N \)-solver against flux-limited diffusion for a set of standard problems in Section 8. Finally, Section 9 concludes with a summary and review of future work.

2. Previous work

Light transport in participating media is governed by the radiative transfer equation (RTE), first studied in the context of astrophysics by Chandrasekhar [Cha60] and later introduced to computer graphics by Kajiya [Kaj86]. Today, this equation is typically solved using Monte Carlo methods. In strongly scattering media or in the presence of highly anisotropic phase functions, these methods can become prohibitively expensive. For example in the case of a high albedo medium like milk where tracing paths with a huge number of scattering events is necessary.

In contrast to path-tracing, the \( P_N \)-method gives a solution by solving a system of linear equations. It is derived by discretizing the angular variable of the radiative transfer equation into spherical harmonics (SH). This gives rise to a set of coupled, complex-valued partial differential equations, called the \( P_N \)-equations. The subscript \( N \) refers to the spherical harmonics truncation order.

The \( P_N \)-method has a long history in other fields and was never applied in graphics. Kajiya [KVH84] explained the theory, but did not give any details on implementation or how to solve it. In fact, as Max [Max95] pointed out, it is not clear if Kajiya succeeded at all at applying the method, as all of the results in his paper were produced with a simpler method. This is further strengthened by our observation that a straightforward finite difference discretization of the \( P_N \)-equations produces unusable results, due to oscillation artifacts in the solution. We derive the real-valued \( P_N \)-equations with a staggered-grid solver, that produces artifact-free solutions.

Similar to \( P_N \), the classical diffusion approximation (CDA) is a deterministic method, which arrives at a solution by solving a system of linear equations. It corresponds to the \( P_N \)-equations when \( N = 1 \) (truncation after the first SH order), which can be collapsed to a simple diffusion equation, giving the method its name. CDA has a long history in other domains, such as astrophysics [Ish78] and was introduced to graphics by Stam [Sta95].

CDA suffers from severe energy loss close to regions with strong density gradients. The problem can be addressed by a modification known as the Variable Eddington factor (VEF) method, which nonlinearly adjusts the diffusion coefficient to improve the solution near density gradients and low-density regions. Flux-limited diffusion, developed in the context of astrophysics by Levermore et al. [LP81] and later introduced to graphics by Koerner et al. [KPS+14], is the most prominent example.

VEF is based on the observation that the behavior of volumetric transport is closely linked to the density of the material: in the diffusive regime, thick highly scattering media is present and causes photons to change directions in short succession, while the transport regime in thin and highly absorbing media causes photons to travel long straight lines. In the absence of scattering (e.g. pure vacuum), the non-linear diffusion coefficient turns the diffusion equation into an advection equation. The idea behind VEF is to derive a better diffusion coefficient by seeing it as a means to spatially blend between the diffusive and pure transport regime.

It is an open and unresolved question whether the \( P_N \)-method with truncation at higher order, gives any benefit over using first order non-linear diffusion. This question has also been raised in other domains [OAH00] and resolving it is one of our motivations.
3. Discretized Radiative Transfer Equation

Our method derives from the radiative transfer equation (RTE), which expresses the change of the radiance field \( L \), with respect to an infinitesimal change of position in direction \( \omega \) at point \( \bar{x} \):

\[
(\nabla \cdot \omega) L(\bar{x}, \omega) = -\sigma_t(\bar{x}) L(\bar{x}, \omega) + \sigma_s(\bar{x}) \int \rho(\omega') L(\bar{x}, \omega') \, d\omega' + Q(\bar{x}, \omega) .
\]

The left hand side (LHS) is the transport term, and we refer to the terms on the right hand side (RHS) as collision, scattering, and source term, respectively. The symbols \( \sigma_t \), \( \sigma_s \), \( \rho \), and \( Q \) refer to the extinction- and scattering coefficient, phase function and emission.

The RTE is often given in operator notation, where transport, collision, and scattering are expressed as operators \( T \), \( C \) and \( S \), which are applied to the radiance field \( L \):

\[
T(L) = -C(L) + S(L) + Q .
\]

Deterministic methods are derived by discretizing the angular and spatial domain. This gives rise to a linear system of equations, which can be solved using standard methods. For the \( P_N \)-method, the angular variable is first discretized, using a truncated spherical harmonics expansion. This results in the \( P_N \)-equations, a system of coupled PDEs that still depend on a continuous spatial variable.

The number of equations grows with the truncation order \( N \). This is why the discretization is laborious and difficult to do without errors if done by hand. We therefore use a computer algebra representation to automate this process. After giving an outline of the general discretization in this section, we will present our computer algebra representation, which allowed us to derive and discretize the \( P_N \)-equations in a semi-automatic fashion.

After substitution, all angular parameters are expressed in terms of spherical harmonics, but they still depend on the continuous angular variable \( \omega \). As a next step, we project each term of the RTE into spherical harmonics, using the projection operator \( P \). This produces a single equation for each \( l, m \)-pair. The \( P_N \)-equations therefore can be written as:

\[
P_l^m T(\hat{L}) = -P_l^m C(\hat{L}) + P_l^m S(\hat{L}) + P_l^m Q .
\]

Once the \( P_N \)-equations have been found, the spatial variable \( \bar{x} \) is discretized using a finite difference (FD) voxel grid (using central differences for differential operators).

Following this discretization, the radiance field \( L \), is represented as a set of SH coefficients per voxel. Flattening these over all voxels into a single vector, gives the solution vector \( \hat{u} \). The RHS vector \( \hat{Q} \) is produced similarly. The projected operators can be expressed as linear transformations, which can be collapsed into a single coefficient matrix \( A \) (see Figure 3):

\[
(T + C - S)\hat{u} = A\hat{u} = \hat{Q} .
\]

\( T, C, S \) are matrices, which result from the discretized transport, collision and scattering operators respectively.

4. Computer Algebra Representation

We use a computer algebra representation for working with the \( P_N \)-equations. It represents the equations using a tree of mathematical expressions, which represent numbers, symbols and other expression types, such as integrals, derivatives, sums, products and functions. Further, manipulators can be executed on these expression trees to perform substitution, constant folding, reordering of nested integrals, application of identities and more complex operations. Finally, frontends allow rendering the expression tree into different forms, such as \texttt{LaTeX} and C++ source code. While we ended up implementing our own lightweight framework, off-the-shelf packages, such as \texttt{SymPy} (www.sympy.org), exist and would be equally suitable for our use case.
Using the computer algebra representation, we perform the derivation steps required to arrive at the real-valued $P_N$-equations (the derivation steps shown in the supplemental material were almost all rendered from the expression tree). More importantly, we use the representation to perform the discretization and generate the stencil code used by our solver. This is detailed in section 6.1.

5. Real-valued $P_N$-Equations

With the help of a computer algebra representation framework, we are able to easily derive and work with the large and unwieldy $P_N$-equations. It is impossible to present the derivations in a meaningful way within the scope of this paper. Displaying a single intermediate step could easily take more than ten A4 pages. We therefore decided to give only the final result in this section, and present the derivation in a supplemental document. We present here the final $P_N$-equations, for a general $N$ and in three dimensions, in a very compact form which we have not found elsewhere in the literature.

Since the real-valued SH bases (Equation 2) have different definitions for $m < 0$, $m = 0$ or $m > 0$, we also get different projections for $S^m$, depending on the sign of $m$.

For $m = 0$ we have

$$\begin{align*}
\frac{1}{\sqrt{2}}E^{l-1}L_{l-1}^{1} & - \frac{1}{\sqrt{2}}E^{l-1}L_{l-1}^{1} + \frac{1}{\sqrt{2}}E^{l-1}L_{l-1}^{1} \\
- \frac{1}{\sqrt{2}}E^{l-1}L_{l}^{1} & + \frac{1}{\sqrt{2}}E^{l-1}L_{l-1}^{1} + \frac{1}{\sqrt{2}}E^{l-1}L_{l}^{1} \\
\sigma^2 L_{2}^{0} - \sigma^2 \lambda^2 p^{2} L_{2}^{0} & = Q_{2}^{0},
\end{align*}$$

(5)

and for $m < 0$ (upper sign) and $m > 0$ (lower sign) we have

$$\begin{align*}
\frac{1}{2}E^{l-1,0}L_{l-1}^{m+1} & - \frac{1}{2}E^{l-1,0}L_{l-1}^{m+1} - \frac{1}{2}E^{l-1,0}L_{l-1}^{m+1} \\
+ \frac{1}{2}E^{l-1,0}L_{l}^{m+1} & + \frac{1}{2}E^{l-1,0}L_{l}^{m+1} + \frac{1}{2}E^{l-1,0}L_{l}^{m+1} \\
\pm \frac{1}{2}E^{l-1,0}L_{l-1}^{m+1} & + \frac{1}{2}E^{l-1,0}L_{l-1}^{m+1} + \frac{1}{2}E^{l-1,0}L_{l-1}^{m+1} \\
\pm \frac{1}{2}E^{l-1,0}L_{l}^{m+1} & + \frac{1}{2}E^{l-1,0}L_{l}^{m+1} + \frac{1}{2}E^{l-1,0}L_{l}^{m+1} \\
\sigma^2 L_{2}^{0} - \sigma^2 \lambda^2 p^{2} L_{2}^{0} & = Q_{2}^{0},
\end{align*}$$

(6)

with

$$\begin{align*}
\beta^m_c = \begin{cases} 
0, & \text{for } m = -1 \\
\frac{1}{2\sqrt{2}}, & \text{for } m \neq -1 \\
\end{cases}, \quad \beta^m_T = \begin{cases} 
2\sqrt{2}, & \text{for } m = -1 \\
0, & \text{for } m \neq -1
\end{cases}
\end{align*}$$

and

$$\begin{align*}
d^m = \sqrt{(l-m+1)(l+m+1) + (2l+1)}/(2l+1) \\
\rho^m = \sqrt{(l-m+1)(l+m)}/(2l+1) \\
\epsilon^m = \sqrt{(l-m+1)(l+m+2)}/(2l+1) \\
\delta^m = \sqrt{(l-m)(l-m-1)}/(2l+1)
\end{align*}$$

$$\lambda_l = \sqrt{\frac{4\pi}{2l+1}}.$$

In the next section we will present the solver that we use to solve the $P_N$-equations.

6. $P_N$-Solver

The truncation order $N$ is the key input parameter to the solver. With higher values, manual implementation of the solver from the equations would be arduous, error-prone and time-consuming. We therefore decided to make use of the computer algebra representation and designed our solver around it.

The solver consists of two components. The first is a precomputation (Section 6.1), which is executed once for every single value of $N$. This step runs a partial evaluation on the mathematical expression tree and applies the spatial discretization in a reference space we call stencil space. The precomputation step automatically generates source code from the resulting expression tree.

The generated stencil code is compiled with the runtime component (Section 6.2) of our solver. This component receives the actual problem as input, including grid resolution and RTE parameter fields. It then builds the linear system and solves it using standard methods. An overview of the solver is given in Figure 5.

6.1. Precomputation

The result of the precomputation is a stencil, which can be used during runtime to build the linear system for a given problem. The stencil is a pattern of indices into the solution vector, along with values. It expresses how the sum of the weighted solution vector components relate to the RHS for a given unknown in the system and therefore contains most information required to fill the system matrix $A$ and RHS-vector $Q$ row by row. Note that while the coefficients may change, the sparsity pattern of the stencil is identical for different rows.
Stencil generation entails discretizing a PDE at a hypothetical center voxel \((i, j, k)\) (assumed to mean the voxel center most of the time). Finite differences create weighted references to other voxels (e.g. \(i+1, j, k\)). After bringing the discretized equation into canonical form (a weighted sum of unknowns), one can write the stencil by reading off the weights and offsets (Figure 6). Voxel \((i, j, k)\) will only be known during runtime, when the stencil is executed for a particular unknown (row). Then the offsets can be used to find the component index into the solution-vector, and weights can be evaluated for concrete world space position. We refer to the space with the hypothetical voxel \((i, j, k)\) at the center as stencil space.

\[
\frac{\partial^2 \phi(x)}{\partial x^2} + \cdots = q(x)
\]

discretization

\[
\frac{\phi_{i+1} - \phi_i}{h} - \frac{\phi_i - \phi_{i-1}}{h} + \cdots = q_{i, j}
\]

canonical form

\[
\frac{1}{h^2} \phi_{i+1} - 2 \frac{1}{h^2} \phi_i + \frac{1}{h^2} \phi_{i-1} + \cdots = q_{i, j}
\]

code generation

Function `applyStencil(A, b, i, j)`

01 function applyStencil(A, b, i, j )
02 row = index(i, j)
03 A[ row, index(i+1, j )] = 1/(h^2 * h)
04 A[ row, index(i, j)] = 2/(h^2 * h)
05 A[ row, index(i-1, j)] = 1/(h^2 * h)
06 ... b[row] = q[i, j]

Figure 6: Creating the stencil code requires several steps, usually done by hand. We express the given problem in a computer algebra representation and use it to fully automate the process.

The spatial discretization is done by parsing the expression tree from the root. The discrete substitute for the continuous position variable \(\vec{x}\) is initialized with \(i, j, k\). Differential operator nodes are replaced by a subtree, which expresses the finite difference approximation (including voxel size factor \(h\)). The subtree of the differential operator node is duplicated for different offsets to the discrete variable \((i, j, k)\). Since this offset only applies to the subtree, a stack is maintained by the parser to manage scope. Whenever the parser encounters the continuous variable \(\vec{x}\) in the expression tree, its node in the tree is replaced by the discrete substitute, currently on top of the stack. Nested differential operators yield higher order finite difference stencils as expected.

Factorization into canonical form is done as a manipulation pass on the mathematical expression tree. The result allows us to implement the code generation in a straightforward fashion. For each term, the \(i, j, k\) offset is retrieved from the unknown. The factor-expression, which is multiplied with the unknown, is extracted from the tree and used to generate source code for evaluating the factor-expression during runtime (including calls for evaluating RTE-parameter fields).

Our solver supports placement of coefficients at arbitrary staggered grid locations (see Figure 7). This means that during the discretization step, the discrete location \((i, j, k)\) (at which the unknown is meant to be evaluated) might not coincide with the location of the unknown. To solve this, depending on how the two are located relative to each
Lseparate the radiance field into single scattered light. We use an approach similar to Koerner et al. [KPS14]. This is done by changing the index of the written component.

rendering BC. This is done by changing the index of the written component. of the coefficient in the closest voxel inside the domain (Neumann
tion (Dirichlet BC) or accumulates into the row and column in
boundary location, the framework either ignores the write opera-
tions. They are handled transparently by the code which generates
the stencil. Whenever the stencil accumulates coefficients into a

tions for boundary conditions

6.2. Runtime
The stencil code is generated once for every value of \( N \) and com-
piled with the runtime component of our solver. The runtime exe-
cutes the stencil for every voxel to populate the system matrix \( A \)
and RHS vector \( \hat{Q} \) with numerical values.

The number of rows is determined by the number of voxels times
the number of coefficients per voxel (see Figure 3) and can there-
fore become very large for high resolution and high truncation or-
der. The matrix \( A \) is square and extremely sparse, due to the local
structure of the finite differences discretization. Unfortunately, it is
non-symmetric due to the transport term and not diagonal domi-
nant, which rules out many iterative methods for solving linear
systems. Iterative methods are useful, as they allow balancing accur-
acy against performance by tweaking the convergence threshold.
We address this by solving the normal form \( A^T A \hat{u} = A^T \hat{Q} \) instead.
This gives a symmetric and positive definite system matrix \( A^T A \),
albeit with a higher condition number. Investigation of other solu-
tion schemes (e.g. multigrid) would be an interesting avenue for fu-
ture work. However, more importantly, in the presence of vacuum
regions, the matrix \( A \) becomes singular and the system cannot be
solved at all. This requires the introduction of a minimum thresh-
hold for the extinction coefficient \( \sigma_t \).

Our solver supports Neumann and Dirichlet boundary condi-
tions. They are handled transparently by the code which generates
the stencil. Whenever the stencil accumulates coefficients into a
boundary location, the framework either ignores the write opera-
ton (Dirichlet BC) or accumulates into the row and column in \( A \)
of the coefficient in the closest voxel inside the domain (Neumann
BC). This is done by changing the index of the written component.

7. Rendering
We use an approach similar to Koerner et al. [KPS14], where we
seperate the radiance field into single scattered light \( L_{ss} \) and multi-
ple scattered light \( L_{ms} \):

\[
L(\vec{x}, \omega) = L_{ss}(\vec{x}, \omega) + L_{ms}(\vec{x}, \omega).
\]

The single scattered light is folded into the emission term \( Q \):

\[
Q(\vec{x}, \omega) = L_{ss}(\vec{x}, \omega) = \sigma_t(\vec{x}) \int_{\Omega} \rho(\omega' \rightarrow \omega) L_u(\vec{x}, \omega') \, d\omega'.
\]

This means that our solver will solve for the multiple scattered
light \( L_{ms} \). The quantity \( L_u \) is the uncollided light, which was emi-
ted from the light source and attenuated by the volume without un-
dergoing any scattering event. We compute it using a few light sam-
ple per voxel, which quickly converge to a useful result for Dirac
delta light sources.

Running the solver gives solution vector \( \hat{u} \). We then un-stag-
ner the solution by interpolating all coefficients to voxel centers. The
additional coefficients at boundary voxels are no longer needed.
This operation is represented as a matrix that produces a three-
dimensional voxel grid with SH coefficients for order \( N \) at the cen-
ter of each voxel.

For rendering, we use a simple forward path tracing approach,
where we start tracing from the camera. At the first scattering event,
we use next event estimation to account for \( L_{ss} \). Then we sample
a new direction according to the phase function. Instead of contin-
uing tracing into the new direction, we evaluate the in-scattering
integral using \( L_{ms} \). The SH coefficients at \( \vec{x} \) are found by trilinear
interpolation from the voxel grid of SH coefficients.

8. Results
In this section, we present results for a set of standard problems in
order to validate and evaluate our method. We also compare against
classical diffusion (CDA) and flux-limited diffusion (FLD).

Our computer algebra framework and the precomputation has
been implemented in Python. The runtime component of our solver
has been implemented in C++. We use a naive implementation of
a CG solver, which has been modified such that we do not need to
explicitly compute the normal form of the linear system to solve.
We use the sparse matrix representation and sparse matrix vector
product from the Eigen linear algebra library (eigen.tuxfamily.org).

The solver for classical diffusion is based on the diffusion equa-
tion, which is derived by collapsing the \( P_1 \)-equations:

\[
\nabla \left( \frac{1}{3\sigma_t} \nabla L^{0.0} \right) = -Q^{0.0}.
\]

Since our solver can work with any PDE which results in a linear
system of equations, we put Equation 9 into our computer algebra
representation and provide it as an input to our solver, which gen-
erates the correct stencil code automatically.

Since FLD is based on a non-linear diffusion equation, we were
not able to use our system in the same way. Our implementation
closely follows the implementation in [KPS14] (though ours runs
on CPU) and we refer to their paper for more details.
8.1. 2D checkerboard

First we ran our solver on the 2D checkerboard, a very common test case in other fields. The problem has dimensions $7 \times 7$ and is discretized with resolution $71 \times 71$. Unit size blocks are filled with purely absorbing medium $\sigma_a = 10$ in a checkerboard pattern. All other blocks are filled with a purely scattering medium with $\sigma_s = 1$.

Solving the standard checkerboard problem, allows us to validate our solver against the solver from Seibold et al. [SF14], which solves for the time-dependent and complex-valued $P_N$-equations in the 2D case only. The 2D case is derived by assuming that all SH coefficients, RTE parameters and boundary conditions are $z$-independent. This causes all SH coefficients and moment equations for which $l + m$ is odd to vanish. Due to the time-dependency, their approach is to do explicit incremental steps in time. We run their solver for many timesteps to get a result close to steady state.

As can be seen in Figure 8, the results from our solver are in good agreement with the results from Seibold et al. [SF14] and verify the correctness of our implementation. Converging to a residual of $10e^{-10}$ takes $0.27s$ for $P_1$ and $25s$ for $P_3$.

8.2. Point source problem

We also run our solver for the point source problem, a single point light in a homogeneous medium. This not only helps to validate our implementation for the 3D case, but also provides information on the accuracy of these methods. We use the Grosjean approximation, which was introduced by D’Eon et al. [dI11] as a very accurate approximation to the ground truth solution.

For our test case, we choose a FD resolution of $80 \times 80 \times 80$, an extinction coefficient $\sigma_t = 8.0$ and albedo $\alpha = 0.9$. We run the solver for different truncation values $N$. In Figure 9 a, we see that the solution becomes increasingly accurate for higher truncation order. The ground truth is underestimated when $N$ is odd, and overestimated when $N$ is even. We further see that the $P_1$ solution exactly matches the results from CDA, which confirms that the latter is only a collapsed version of the former. The time to solve is significant with $10m$ for $P_1$ and $45m$ with $P_3$. With these performance characteristics, our $P_N$-solver is clearly not competitive in comparison with CDA and FLD solver, which are much faster.

8.3. Nebulae

Finally, we run our solver on a procedural cloud dataset to get an idea of its performance in more practical applications. Figure 1b shows the result of $P_5$ for a procedurally generated heterogeneous cloud with an isotropic phase function. We see that at order $N = 5$, our method can capture indirect illumination similarly well as FLD and is significantly better than CDA as expected. The indirectly illuminated region at the bottom appears to be closer to the path-traced result as opposed to the solution from FLD which is very flat in that region. However, in many other areas, $P_3$ seems to still suffer a lot from the problem of energy loss near transitions from dense regions to vacuum. It appears that going higher order a few steps mitigates this only slowly at a high cost of compute and storage.

The main characteristic of the nebulae dataset is the presence of vacuum. We found that having vacuum regions in the dataset will cause the condition number to become infinite and the solver practically does not converge. We therefore introduced a minimum threshold for the extinction coefficient $\sigma_t$. Every voxel with an extinction coefficient smaller than the threshold is set to the threshold value. In Figure 10 we show the effect of the minimum threshold on the convergence behavior. As it increases, convergence improves.

9. Conclusion

In this paper, we introduced the $P_N$-method to the toolbox of deterministic methods for rendering participating media in computer graphics. We derived and presented the real-valued $P_N$-equations, along with a staggered grid solver for solving them. We showed how to use the results in a rendering system and ran our solver for various standard problems for validation.

We originally set out to understand how non-linear diffusion compares to $P_N$ for increasing order. Although the lack of higher moments makes the FLD solution very flat, its energy conserving nature and comparably small resource footprint make it a much better approach when compared to $P_N$, which becomes increasingly costly for higher values of $N$. 

Figure 8: Comparison of the result for the checkerboard test using StaRMAP’s time-stepping solver [SF14] (left) against our steady-state solver (right) with $N = 5$. Our results are in good agreement.

Figure 9: Lineplot through the 3D solution of our solver for the point source problem for various order $N$ (left). Solution for $P_5$ compared against classical diffusion, flux-limited diffusion and analytical solution (right).

Figure 10: Convergence behavior (a) of $P_N$ as a function of vacuum size (left) and $\sigma_t$ (right).
therefore can generate stencil code for all these variations by sim-
artifacts, dealing with vacuum regions and general convergence.
dress certain problems of the standard P (Matlabs condest function) are shown next to the plots. The convergence deteriorates as the threshold decreases. Once it reaches zero, the presence of pure vacuum makes the condition number infinite.

The literature in other fields often states that the PN method—when solved in normal form like we do—is able to deal with vacuum regions. We found this misleading. The PN-method in normal form indeed does not break down completely in the presence of vacuum as diffusion based methods do (due to $\sigma_t^{-1}$ in the diffusion coefficient). However, in the presence of vacuum, the condition number of the system matrix becomes infinite and does not converge either. Therefore PN based methods also require minimum thresholding of the extinction coefficient and offer no benefit for vacuum regions.

Much more work needs to be done in order to make the PN method competitive in performance to alternative solutions for vol-
ume rendering. We believe this can be made possible by employ-
ing a multigrid scheme for solving the linear system of equations. We implemented a multigrid solver, but did not find the expected performance improvements. This is possibly due to the coupling between coefficients within a voxel, which does not work well to-
gether with the smoothing step. We want to study this in the future.

Unique to our system is that it uses a computer algebra represen-
tation of the equation to solve as input. Discretization in angular and spatial domain is done using manipulation passes on the repre-
sentation and providing this as an input to our solver. This would allow an exhaustive comparison of all these methods and we con-
sider this as future work.

Finally, since the approach of our solver is very generic, we also would like to explore its application to other simulation problems in computer graphics.

Acknowledgements
We thank the anonymous reviewers for their valuable and encour-
aging comments and feedback. We also thank Martin Frank and Benjamin Seibold for the very valuable discussions and answers to our questions. We thank Bernd Eberhardt for feedback and support. This project has been partially funded by the MSC-BW project.

References


[dI11] d’EON E., IRVING G.: A quantized-diffusion model for render-
ing translucent materials. ACM TOG (Proc. of SIGGRAPH) 30, 4 (July

Y.: A least-squares transport equation compatible with voids. Journal of

[Ish78] ISHIMARU A.: Wave propagation and scattering in random me-
dia. single scattering and transport theory. 2


[KVH84] KAIHYA J. T., VON HERZEN B. P.: Ray tracing volume densi-
174.

[LP81] LEVERMORE C. D., POMRANING G. C.: A flux-limited diffu-

[Mas95] MAX N.: Efficient Light Propagation for Multiple Anisotropic


Spectroscopy and Radiative Transfer 64, 6 (2000), 619 – 634.


[SFL11] SCHAFER M., FRANK M., LEVERMORE C. D.: Diffusive cor-
rrections to pn approximations. Multiscale Modeling and Simulation 9