柔軟性のある物理ベースレンダリングアーキテクチャの開発

上野康平 *

Unified Rendering Architecture

Kouhei Ueno

概 要

最近、計算機の性能が高まるにつれて、物理学により即した光学シミュレーションを行う物理ベースレンダリングが注目を浴びている。非常に写実的な画像が生成できるのが特徴であるが、逆にトゥーンシェーディングに代表されるようなアート系の表現を不得意とする、微調整が困難、計算時間が増えるといった欠点がある。そこで本稿では、これらの問題を解決する統合型レンダリングアーキテクチャ、物理ベースシェーダシステム (PBS)、及びレンダラ特化分散計算ライブラリ (libpolatsk) を提案する。

Abstract

We present a new rendering architecture enabling more flexible physically-based rendering. This architecture, called "unified rendering architecture" combines previous appearance-based rendering techniques with recent physically-based rendering techniques. The problem with previous physically-based renderer is that it lacks support for artistic expressions, such as toon shading, and has poor ability for result image tuning. We have solved those issues by introducing a programmable shader system into physically-based rendering, which we call PBS, physically-based shader. We have also developed a distributed computing library, libpolatsk, which has been exclusively optimized for rendering applications.

1 はじめに

近年、レンダリングアルゴリズムは急速な進歩を遂げている。Kajiya らによって物理ベース CG の基盤となるレンダリング方程式 [1] が提唱されてから二十年、物理学的に正確な理論に基づいたレンダリングアルゴリズムは実用の域に達してきた。しかし、実際に現在プロダクションレベルで使われているのは、経験論に基づいた数式を用いる古典的な手法が殆どである。なぜなら、既存の物理ベースのレンダラは、写実的な画像は正確に計算できても、特殊効果等に使われる複雑で非現実的な材質や、影の強調や間接照明の調整といった誇張表現は難しいといった欠点があるためである。

そこで、3DCG をレンダリングするための新しい統合型アーキテクチャを設計した。照度計算に物理ベースアルゴリズム、特殊効果処理に古典的アルゴリズムと二つのアルゴリズムの併用を可能にすることにより、物理シミュレーションに基づいた写実的な画像、特殊効果のかかったアーティスティックな画像の両方を同じソフトウェアで扱うことを実現した。

また、上記の統合型アーキテクチャで使用できるシェーダシステム、PBS（Physically-based Shader）を設計、開発した。シェーダをツリー状の UI でビジュアルプログラミングすることにより、システムを意識することなく他レンダラと変わらぬ操作感で材質表現に集中できる環境を整えた。

さらに、レンダリングの計算特性を考慮し、分散レンダリングを効率的に行うことのできるタスクベースの計算分散システムを設計、実装した。非対称かつ低信頼性の PC で構成されるネットワークでも効率的に分散レンダリングを行うことを可能にした。

*千葉大学 k-ueno@cfs.chiba-u.ac.jp

2 関連研究

2.1 物理ベースレンダリング

レンダリング計算を形式的に記述した「レンダリング方程式」[1] が Kajiya によって提唱された後、Greenberg らのレンダリング理論に物理学的根拠を求める強い主張 [2] を経て、BRDF (Bidirectional Reflectance Distribution Function) 関数のエネルギー保存則適合化 [3] や Spectral Rendering[4] 等が提唱された。

本研究では、大域照明計算を上の主張のように、合理的な範囲で光学理論に沿って行う。

2.2 レンダリングアーキテクチャ

古典的レンダリング（非物理ベースレンダリング）を行うシステムの代表として、REYES (Renders Everything You Ever Saw) アーキテクチャ[5] が挙げられる。米 Pixar 社のレンダラ、RenderMan が採用しており、シェーダと呼ばれるスクリプトを使用し、高い自由度でマテリアル (材質の反射特性) を記述することができる。一方、物理ベースレンダリングを行うアーキテクチャとしては、RADIANCE[6] が有名である。また、Vision システム [7] はオブジェクト指向プログラミングを活用し、大域照明エンジンの複数実装に成功している。

2.3 シェーダ

シェーダの効率的記述については、Shade Tree[8] が有名である。今まで通常のプログラムの様にテキストで表現されていたシェーダを、有向非循環グラフ (DAG) 状に記述することで、シェーダの可読性、部品の再利用性を上げようという試みである。いくつかの 3D モデリングソフトに搭載されているが、レンダラには通常のテキスト状のプログラムにコンパイルされて渡され、あくまでシェーダデザイン UI として用いられている。

RTSL(Ray Tracing Shading Language)[9] はレイトレーシング向けのシェーディング言語を作る試みである。リアルタイムレイトレーシングエンジン Manta に実装されている。PBS とは異なり、C++のようなテキストベースのオブジェクト指向言語である。しかし、材質の物理的な反射特性として記述されている部分と特殊効果エフェクトとして記述している部分は分離できていない。よって、厳密な物理ベースレンダリングは困難である。

2.4 分散レンダリング

レンダリングは、非常に多くの計算量を必要とするため、複数のコンピュータで分散計算することが考えだされた。本研究では、レイトレーシング法のアルゴリズムのレベルでの分散計算に成功した Kilauea レンダラ [10] の手法を参考にした。

3 統合型レンダリングアーキテクチャ

統合型レンダリングアーキテクチャは、物理ベースレンダラをベースに、古典的レンダラの持つ様々な表現の多様性を取り入れたものである。

物理レンダラをベースにする理由はいくつかあるが、最大の理由は計算の安定性である。古典レンダラの計算式はエネルギー保存則を満たさないものも多く、計算が発散してしまうことが多々ある。物理レンダラを基盤にし、放射照度計算をすべて物理レンダラ側に任せてしまうことで、この問題を避けることができる。

まず、統合型レンダリングアーキテクチャの設計について述べる。

物理ベースレンダラだけでは、物理法則に逆らうような特殊効果や誇張表現をうまく実現できない。そこで、エフェクト部分の処理を古典的レンダラベースのエフェクタとして分離した。これにより、計算の安定性、生成画像の写実性、表現の多様性をすべて実現することができる。

図 1 は従来のアーキテクチャと今回開発したアーキテクチャを比較したブロック図である。既存レンダラとは違い、提案システムでは三種類のレンダリングパスが構成可能である。

A. エフェクタ部のみを通るパス

古典的レンダラとして振る舞う。

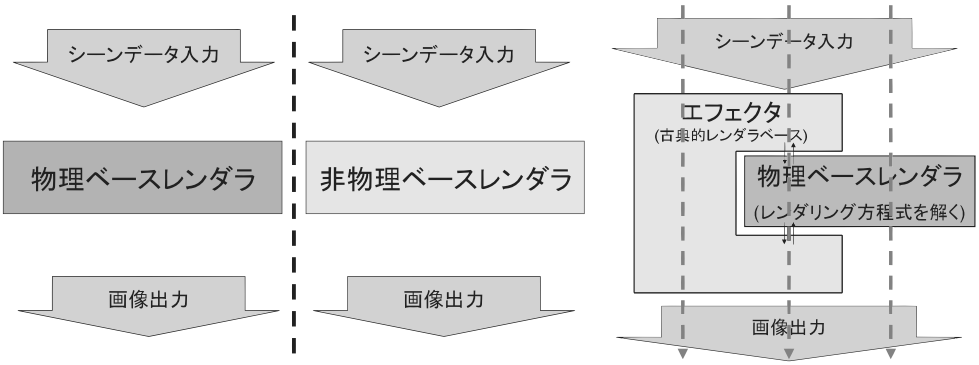

図 1: 既存レンダラ／提案レンダラ　比較ブロック図

エフェクタ部は古典的レンダラをベースに拡張したものとなっており、擬似照明計算ですませるシーンはエフェクト部のみでレンダリングすることが可能となっている。エフェクタ部用シェーダ実行環境は古典的レンダラと同じ高い自由度を実現しており、幅広い表現が可能となっている。

B. 物理レンダラ部のみを通るパス

物理ベースレンダラとして振る舞う。

物理レンダラ部は正確な物理シミュレーションを行い、厳密な証明計算を行う。古典的レンダラでの大域照明 (GI) アルゴリズムと違い、計算に一切ごまかしを入れないので、精度の高い結果が期待できる。

このパスでは、そのシミュレーション結果を直接出力することによって、物理ベースレンダラの写実性をそのまま活かした画像がレンダリングできる。

C. 物理レンダラ・エフェクタ両方を通るパス

物理ベースレンダラと古典的レンダラで分担処理を行う。

エフェクタ部のシェーダ実行環境には、物理ベースレンダラ部に照明計算を依頼するインターフェースが実装されている。これにより、エフェクト処理を古典レンダラに、照明計算を物理ベースレンダラに分担処理させることが可能となる。

照明計算が厳密に行えるので、実写に特殊効果を合成するような用途で非常に威力を発揮する。その他のシーンでも、物理ベースレンダリング向けの強力な大域照明 (GI) アルゴリズムを活用できるというメリットがある。

また、後述する PBS とあわせて使用することで、この分担処理を意識することなく、古典的レンダラと同じ使い勝手でこのシステムの利点を享受することが可能となっている。

4　物理ベースシェーダ: PBS

古典的レンダラには、複雑な材質を表現するのに「シェーダ」と呼ばれるスクリプトが使われる。これにより、自由に材質の見た目が表現できるようになっている。しかし、これを前述の統合型アーキテクチャにそのまま適用しようとすると、様々な問題が生じる。

まず、古典的レンダラでは視点方向からの応答のみを記述すれば十分であったのに対して、統合型レンダリングアーキテクチャは以下に挙げた 4 通りの反射特性定義を必要とする。

視点方向応答
: 視点方向からのレイに対する応答関数。古典的レンダリングではこの関数のみが使用される。

BSDF[1]関数定義
: 双方向拡散分布関数 $f_s(x, \Theta \leftrightarrow \Psi)$。物理ベースレンダリングに用いられる。

BSDF サンプリング用サンプラー
: 双方向拡散分布関数用サンプリング関数。インポー

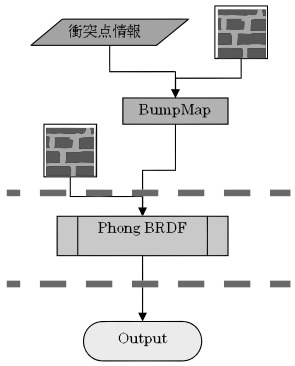

図 2: シェードツリー例

タンスサンプリングに用いられる。以下のような確率分布 p_b を持つ:

$$p_b(\vec{\omega}_\Psi)\big|_{x,\omega_\Theta} = \frac{f_s(x,\Theta \leftrightarrow \Psi)}{\int f_s(x,\Theta \leftrightarrow \Psi)\mathrm{d}\vec{\omega}_\Psi}$$

光源方向応答

光源方向からのレイに対する応答関数。フォトンマッピング等の大域照明アルゴリズムで用いられる。

また、統合型アーキテクチャでは照明計算部とエフェクト部が完全に分離しているため、シェーダプログラムも材質の反射特性定義とエフェクト定義に適宜分離する必要がある。

これらの問題を、PBS ではシェードツリーをそのままシェーダのプログラミングに活用することで解決している。(図 2) ツリー状の UI でシェーダをビジュアルプログラミングすることによって、統合型アーキテクチャの特殊なパイプラインを意識することなく、シェーダプログラムを記述することが可能になっている。シェードツリーを構成するノードの種類は、以下の 3 種類に分けられる:

テクスチャ合成ノード テクスチャ合成を行うノード。エフェクタ部で処理される。

光積分ノード Phong モデル、Lafortune モデルといった物体の BRDF モデルに基づき、光積分を行うノード。照度計算部で処理される。

[1]Bidirectional Scattering Distribution Function(双方向拡散分布関数)

エフェクタノード 光積分ノードからの結果に対しポストエフェクトを加えるノード。エフェクタ部で処理される。

このようにツリー状に記述されたシェーダから、エフェクト部、照度計算部を抽出するのは各ノードの種類の判別を行うだけで済む。

PBS では、他のレンダラとは違い、ツリー状のシェーダプログラムを直接読み込む。各ノードは、クラスとして実装されており、レンダラ内ではそれらのポインタツリーとして保持される。ノード実装クラスには、先に述べた 4 種類の関数定義に対する応答が実装されている。

このような実装にすることによって、ユーザはシェーダを書く度に 4 種類の定義を書く必要がなくなる。光積分ノードの実装を書くときのみ、冗長な実装を必要とするが、一般的なシェーダではこれらは共通であるため、ユーザがこれを書く必要に迫られることは殆どない。

また、シェーダツリーの構造は有向非循環グラフ (DAG) に限定されるため、スクリプト言語では記述できてしまう照度計算結果をテクスチャに再代入するといった意味をなさないシェーダを制限できるというメリットがある。

5 分散計算ライブラリ: libpolatsk

厳密なシミュレーションを行う物理ベースレンダリングは、莫大な計算量を要する。従って、PC クラスタを使って分散計算を行うのが合理的である。今回、統合型レンダリングアーキテクチャの特徴である:

- 極めて細かいタスク分割
- 引数、結果データが小さい (1 つの UDP メッセージに収まる)

を活かし、効率的な分散レンダリングを行うために、分散計算ライブラリ libpolatsk を新規に開発した。

libpolatsk はタスクシステムを基盤にした分散計算ライブラリである。ユーザーからは、ネットワーク上に一つのタスクプールがあるように見え、そこにタスクを投げ込むことで最適なノードに自動的に転送され、実行される。

タスクの処理ノード選択には、タスクの粒度に応じて二種類のアルゴリズムを使い分けている。粒度が細かいタスクに対しては、単純にノード性能で重み付けをした乱数によって決定するアルゴリズムを採用し、粒度が粗いタスクに対しては、実行時間予測によるヒューリスティクスを使ったアルゴリズムを採用している。粒度が細かいタスクは、処理時間が短いため、予測が仮に失敗しても問題は少ない。それよりも予測自体の高速性が求められる。これに対し、粒度の粗いタスクは、処理時間が長いため、多少時間をかけても正確な予測を行う必要がある。二種類のアルゴリズムを用意することで、両方のケースに対応している。

タスクシステムの実装にあたり、Coroutine（ユーザレベルスレッド）の実装が必要になった。短期間で可搬性のある実装を書くのは困難と判断し、Io 言語 [11] の VM から該当部分をライブラリとして抽出し、C++ 言語のラッパーを書いた[2]。

また、libpolatsk は非対称な動的ネットワーク環境に対応している。これは、低信頼性のノードでレンダーファームが構成された場合を考慮している。PC の高性能化、低価格化の傾向の中、こういった環境に対応することは重要であると考えられる。

ノード間メッセージ通信には UDP/IP 通信を使用している。UDP は、セッションを張る必要がなく、高速な通信ができるのが特徴である。また、ブロードキャストをサポートしており、一対多のメッセージ配信を行う際には、TCP に比べ圧倒的なパフォーマンスが得られる。UDP はパケットの順序保存や送達保障をプロトコルレベルでサポートしていないという点で TCP に劣るが、順序保存に関しては、レンダリング計算のタスク要求は全て UDP メッセージサイズの 512 バイト以内に収まる為、問題は生じない。送達保障については、libpolatsk の通信部で実装をしている。また、シーンデータやテクスチャ画像データ等、UDP メッセージに収まりきらないデータ通信は、TCP によって行っている。

6 プロトタイプ実装: nytr

上記の統合アーキテクチャ、PBS システム、分散計算システムを実装したレンダラ “nytr ” を開発した。

[2] http://websvn.nyaxtstep.com/viewvc.cgi/3rdparty/libcoroutine/

開発には、C++言語を用いた。Microsoft Windows XP, Linux, Mac OS X の主要プラットフォームで動作する。可搬性が高いコードになっているため、他プラットフォームへの移植も容易である。

現在、nytr レンダラは GNU Public License の下、フリーで公開している[3]。

6.1 機能

一般的な商用レンダラと同程度の機能は実装した。機能一覧は長いため、レンダリングアルゴリズムのみを下に記す。

古典的レンダリングアルゴリズム:

- Traditional Ray Tracing (Whitted Ray Tracing)

物理ベースレンダリングアルゴリズム／大域照明 (GI) 計算アルゴリズム:

- Monte-Carlo Path Tracing
- Bidirectional Path Tracing
- Photon Mapping
- Final Gathering
- Irradiance Caching (with gradient)

6.2 拡張性・動的構成機能

nytr レンダラの特徴として、上記技術以外に、その拡張性、動的構成機能があげられる。レイ交差判定部、照度計算部（物理ベースレンダラ部）、エフェクタ部（古典的レンダラ部）は独立しており、実行時に動的に構成することが可能になっている。

例えば、交差判定部を標準の三角形ポリゴンモデル専用のものから height-map の交差判定と置き換えることで数値シミュレーションや測地データの可視化等の用途に使うことも可能である。

物理レンダラベースの照度計算部は特に柔軟な実装が可能となっている。直接光照度計算部、間接光照度計算部、パストレーシングコア部、フィルタ部でモジュールが独立しており、これらも実行時構成が可能である。これを特に活用しているのが放射照度キャッシュやファ

[3] http://nyaxtstep.com/projects/nytr

イナルギャザリングといったキャッシュ・フィルター系の実装で、末端のアルゴリズムに依存することなく実装されている。

実践的な使用でも、作品のシーン特性によって効果的なアルゴリズムを適宜選択することで、より高品質な画像を得ることができる。

6.3 複数チャンネル出力

一般的に古典的レンダリングアルゴリズムに比べて、物理ベースレンダリングアルゴリズムは、多くの計算時間を必要とする。nytr では、レンダリング後調整の自由度を上げるため、レンダリング結果画像に様々な情報を付加することができる。以下にその活用例を挙げた：

深度 疑似被写界深度表現や、他 3D 画像との合成等

オブジェクト ID/マテリアル ID 特定オブジェクトに対する 2D ポストエフェクト

法線ベクトル 後付けハイライト効果等

光源別チャンネル ライティング強度のレンダリング後調整

直接光/間接光分離 間接光成分を強調することにより結果画像に暖かみを出す等

7 評価

7.1 統合型アーキテクチャ及び PBS

統合型レンダリングアーキテクチャ及び PBS を実装したプロトタイプレンダラ nytr のレンダリング例を示す。(図 4) 同じ缶のモデルを 3 種類のパイプラインを使用しレンダリングしている。左右の画像はエフェクタ部、物理ベース照明計算部を独立に用いたのに対し、中央の画像では物理ベース照明計算部による出力にエフェクトを加えている。

物理ベースレンダリングの精密さを活かしつつ、特殊効果エフェクトが加えられていることがわかる。

[4]モデルデータは http://www.3dnuts.com/ より引用

7.2 分散計算ライブラリ: libpolatsk

分散 RPC ライブラリである OmniRPC[5] とパフォーマンス比較を行った結果 (表 1)、リモートノード上の sin 関数の平均呼び出し時間は libpolatsk の方が約 1.38 倍早いことがわかる。しかも、この結果は OmniRPC が rsh コネクションを使いまわせた場合の結果であるから、大量のノードに向かってメッセージを送信するような実際の計算ではさらに差がつくと考えられる。

また、libpolatsk では、動的ネットワークもサポートしている。インターネット環境向けグリッドコンピューティングツールキットには、一部搭載されているものもあるが、libpolatsk では LAN 環境を前提としているという点が異なる。認証やデータベース等、セットアップに手間取ることなく、初期ノード情報のみ設定するだけで簡単にネットワーク構築ができる。比較の為にいくつかの分散計算システムを調べてみたところ、libpolatsk では設定ファイルなしに他ノードのアドレス一つを与えるだけで自動設定できるところを、OmniRPC では XML 設定ファイルを記述する必要があり、Ninf-G[6]では認証サーバ・SQL サーバを立ち上げる必要があった。

8 まとめ

本稿では古典的レンダリング、物理ベースレンダリングを組み合わせた統合型レンダリングアーキテクチャ、シェーダ技術 PBS が有用であることを示した。統合型レンダリングアーキテクチャを利用することで、物理ベースレンダリングの強力な写実性をそのままに、古典的レンダラの特徴であった柔軟な表現が可能になる。

また、レンダリング計算に特化した分散計算ライブラリ libpolatsk について述べた。libpolatsk は、粒度の細かいタスクについて最適化されており、非対称なノードによって構成される動的なネットワーク環境で動作する。

[5]http://www.omni.hpcc.jp/OmniRPC/
[6]http://ninf.apgrid.org

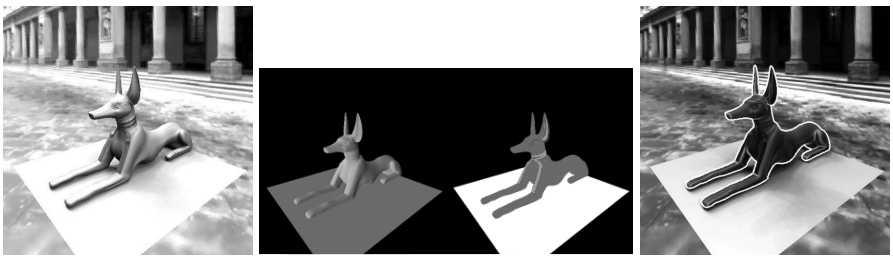

図 3: nytr マルチチャンネルレンダリング例

(左:オリジナル画像　中央:法線ベクトル、マテリアル ID 出力　右：コンポジット例)[4]

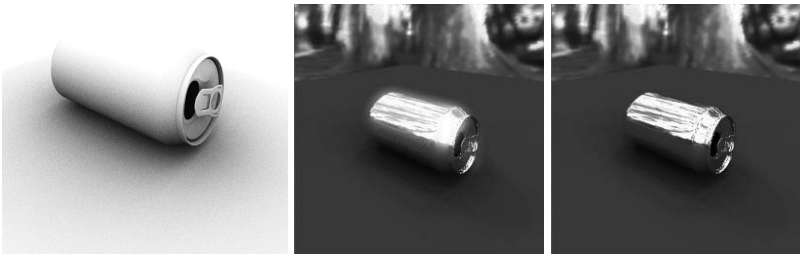

図 4: nytr パイプライン別レンダリング例

(左:エフェクタ部のみ使用　中央:物理ベースレンダラ＋エフェクタ　右：物理ベースレンダラ出力のみ)

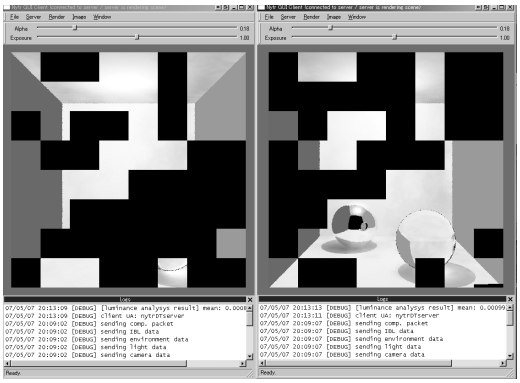

図 5: libpolatsk による分散レンダリング例

	試行 1	試行 2	試行 3	平均
libpolatsk	381msec	392msec	373msec	382msec
OmniRPC	532msec	524msec	528msec	528msec

表 1: libpolatsk 性能比較

9 謝辞

本研究の一部は、独立行政法人　情報推進機構 (IPA) の 2006 年度下期未踏ソフトウェア創造事業（未踏ユース）の支援を受けて行われた。プロジェクトマネージャを務めて下さった東京大学大学院の竹内郁雄教授には、特にお世話になった。この場を借りて深く感謝する。

参考文献

[1] James T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pp. 143–150, New York, NY, USA, 1986. ACM.

[2] Donald P. Greenberg, Kenneth E. Torrance, Peter Shirley, James Arvo, Eric Lafortune, James A. Ferwerda, Bruce Walter, Ben Trumbore, Sumanta Pattanaik, and Sing-Choong Foo. A framework for realistic image synthesis. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 477–494, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[3] E.P. Lafortune and Y.D. Willems. Using the modified phong reflectance model for physically based rendering. *Report CW*, Vol. 197, , 1994.

[4] K. Devlin, A. Chalmers, A. Wilkie, and W. Purgathofer. Tone Reproduction and Physically Based Spectral Rendering. *Eurographics 2002: State of the Art Reports*, pp. 101–123, 2002.

[5] R.L. Cook, L. Carpenter, and E. Catmull. The Reyes image rendering architecture. *ACM SIGGRAPH Computer Graphics*, Vol. 21, No. 4, pp. 95–102, 1987.

[6] Gregory J. Ward. The radiance lighting simulation and rendering system. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 459–472, New York, NY, USA, 1994. ACM.

[7] P. Slusallek. *Vision-an Architecture for physically-based Rendering*. PhD thesis, Inst. für Math. Maschinen und Datenverarbeitung (Informatik), 1996.

[8] Robert L. Cook. Shade trees. *SIGGRAPH Comput. Graph.*, Vol. 18, No. 3, pp. 223–231, 1984.

[9] S.G. Parker, S. Boulos, J. Bigler, and A. Robison. RTSL: a Ray Tracing Shading Language. *Interactive Ray Tracing, 2007. RT'07. IEEE Symposium on*, pp. 149–160, 2007.

[10] T. Kato. Kilauea—parallel global illumination renderer. *Parallel Computing*, Vol. 29, No. 3, pp. 289–310, 2003.

[11] Steve Dekorte. Io Language. http://www.iolanguage.com.