グラフィクスとCAD 56－2
（1992. 5. 22）

メッセージ駆動型シェーディングモデル作成支援システム

松井 一樹† 大野 義夫
慶應義塾大学 理工学部

これまでCGの分野では、よりリアルな画像や芸術性の高い画像を計算機を使って生成するために、さまざまなシェーディングモデルが開発されてきた。従来のレンダリングパッケージでは、シェーディングには単一の複雑な計算式を使用して、その計算式中の固有なパラメータを変えることで、それぞれのシェーディングモデルを設計していたが、shade treeの概念によって、その後のレンダリングパッケージに於けるシェーディングモデルの設計が柔軟なものになった。そこで、本研究では、このシェーディングモデルを専門知識を特に持たないユーザーでも作成できるように、オブジェクト指向モデルの概念の基に設計するシステムを提案する。また本システムでは、クライアント・サーバー分散モデルを採用し、シェーディングに関するノウハウをうまくデータベース化できるような枠組を用いている。

A MESSAGE DRIVEN SYSTEM FOR CONSTRUCTING SHADING MODELS

Kazuki Matsui† Yoshio Ohno
Keio University, Faculty of Science and Technology

In computer graphics, large variety of shading models have been developed. How to design an interface to construct new complex shading models simply and effectively from the existing shading models is now important. In this article, a system which allows users to make these shading models more flexibly and intuitively is presented. A lot of special knowledge and training are required to construct shading models. In this system, even a user who has little knowledge on shading algorithms can make good shading models. In addition, know-how on making shading models can be preserved in the database.

†現在の所属: 富士通研究所 (Fujitsu Lab.)

1 はじめに

これまでCGの分野では、よりリアルな画像や芸術性の高い画像を計算機を使って生成するために、さまざまなシェーディングモデルが開発されてきた。これらのシェーディングモデルを一般的に扱うために、1984年、Robert L. Cookはshade treeというコンセプトを発表した。このshade treeによって、レンダリングパッケージの中で、さまざまなシェーディングモデルが目的に応じて設計できるようになった。

従来のレンダリングパッケージでは、シェーディングには単一の複雑な計算式を使用して、その計算式中の固有なパラメータを変えることで、それぞれのシェーディングモデルを設計していたが、このshade treeのアプローチによって、その後のレンダリングパッケージに於けるシェーディングモデルの設計が柔軟なものになった。

しかし実際には、あるシェーディングモデルを自分で作ることは、そのシェーディングモデルが複雑になるにつれて困難になる。つまり、shade treeを作るということは、シェーディングのアルゴリズムを組み立てることであり、shade treeによって、あるシェーディングモデルを設計したとしても、適切なシェーディングパラメータを与えなければならない。

より高品質な画像を生成する為にも、シェーディングモデルの開発は必要であり、それと同時に、それらのシェーディングモデルを用いて、如何に簡単に効率良く複雑なシェーディングモデルを作ることが出来るインターフェイスを設計するかが問題になってきている。

そこで、本研究では、このシェーディングモデルを専門知識を特に持たないユーザーでも作成できるように、オブジェクト指向モデルの概念の基に設計したシステムを提案する。また本システムでは、クライアント・サーバー分散モデルを採用し、シェーディングに関するノウハウをうまくデータベース化できるような枠組を用いている。

2 シェーディングモデルの設計

2.1 シェーディングモデル

1980年に、Turner Whittedが照明に関する論文[Whitted 80]を発表して以来、CGに於けるレンダリングの技法の1つとして、レイトレーシングがもっとも一般的になった。レイトレーシングによって、物体の反射光、屈折光などがうまく扱えるようになり、それと共に、生成される画像の品質も向上し、よりリアルになった。図1に、レイトレーシングでの光線の進み方の簡単な例が示してある。ここで、光線Rは反射光、Tは屈折光、

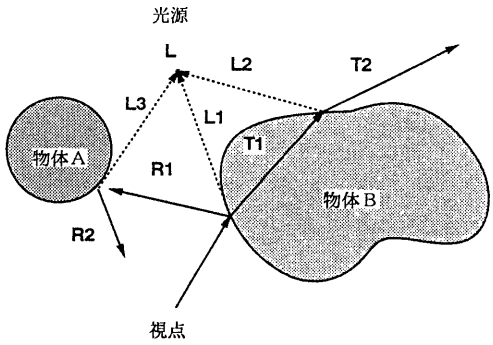

図1: Ray tracing

Lは光源への方向ベクトルである。

照明モデルが作られると、今度はさまざまな物体の見え方を定義する必要が生じた。例えば、よりリアルな木や、石や、その他の物質をCGで作るための、シェーディングモデルが必要になった。

これまでCGの分野では、必要に応じてさまざまな物質のシェーディングモデルが開発されてきた。また、色や表面の状態だけでなく、木や葉自体の形などの形状モデリングに関する研究も盛んに行なわれてきた。その為、それぞれの物質には、それぞれ違ったシェーディングモデルが存在する。このことは、CGのレンダリング、あるいはシェーディングの本質的な難しさである。例えば、ある風景をCGで作成する場合に、その中に含まれるいろんな木や建物、人物や背景などに対して、それぞれ違ったレンダリングを行なわなければならない。

2.2 Shade Trees

1984年、Robert L.Cookはシェーディングモデルの設計のアプローチとして、**shade trees**[Cook 84]を示した。図2にshade treeの例を示す。

このshade treeは大域的な物体の幾何情報にあまり依存しないシェーディングモデルである。つまり局所照明モデルであり、シェーディングの計算には物体と光線の交点情報や、その交点での法線ベクトルなどの局所的な情報が使われる。

このshade treeには、同様に光源によるシェーディングを決める**light tree**や、環境光などのシェーディングを決める**atmosphere tree**が定義されている。そして、木を接木するように、予め作られた小さなshade treeを組み合わせて、さらに大きなshade treeを作ってい

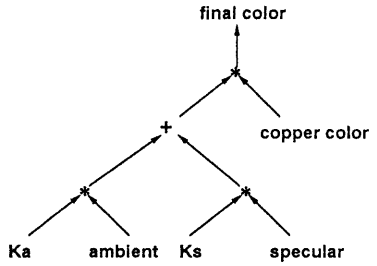

図 2: Shade trees

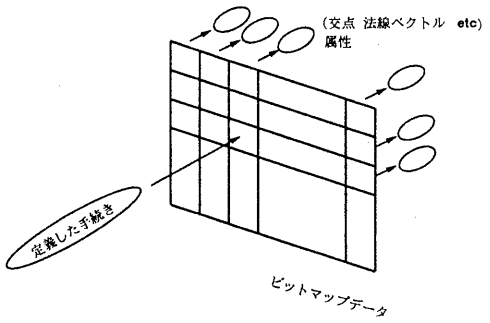

図 3: Pixel stream editor

く。これらの shade tree は、同時に開発されたシェーディング言語を用いて記述でき、それをコンパイルすることで、レンダリングシステムに組み込まれる。

shade tree の概念は、「自分で使うシェーディングモデルは自分で作る」、謂わば、「do it your self」である。この shade tree の概念は、後に Pixar が開発した **RenderMan**[Pixar 88] によって実現されている。

この shade tree によって、シェーディングの過程がきれいに整理されたが、これによってシェーディングモデル設計の本質的な難しさがなくなった訳ではない。つまり、shade tree を組み立てるということは、シェーディングのアルゴリズムを組み立てることと同じことであり、ある程度シェーディングのアルゴリズムを見通す専門知識や、ある程度の経験が必要であろう。

2.3 Pixel Stream Editor

シェーディングモデルの設計について、shade tree とは別のアプローチとして、Ken Perlin は **PSE** (Pixel Strem Editor)[Perlin 85] を開発した。

シェーディングモデルの設計という観点では、図 3に示したように、Perlin は画像データに、物体の交点や法線ベクトルなどの付加情報を含ませて、予め作った小さなプログラムをそれぞれの画素に対して適用して、シェーディングを後処理として行ない、その結果としてその画素での色を決定する方法を使用した。この中で、シェーディング言語として、C 言語のようなものを使っている。

PSE では、一度レンダリングをして画像を生成した後に、その画像に付加情報 (特に、幾何情報) を加えることで、その付加情報を使って何度でも、異なるシェーディングが適応できる点では効率的である。但し、この PSE ではあくまでシェーディングの簡単な実験しか行なえず、レイトレーシングなどのレンダリングが出来ないなどの問題点があり、実際のレンダリングパッケージには適応できない。

また、PSE ではシェーディング言語を用いてシェーディングモデルを設計するが、ユーザーのプログラミング、コンパイリング、実行、デバッギングのサイクルのそれぞれの過程を小さくしたに過ぎず、ここでもシェーディングモデルの設計の困難さはそのまま残ってしまう。

2.4 レンダリング過程の分類

RenderMan の中では、レンダリングの過程は大きく分けると、ライトソースシェーダー (light source shader)、サーフェイスシェーダー (surface shader)、ボリュームシェーダー (volume shader) の 3 種類がある。

図 2.4に於いて、シェーダー (shader) とはオブジェクト指向モデルでのフレーバー (flavor) であり、インスタンス (instance) である。このシェーダーは RenderMan で提供されているシェーディング言語を用いて記述する。このサーフェイスシェーダーは、物体の局所的な幾何情報を利用している。

このように、レンダリングの過程を分類して、分業化することで、固定された 1 つの計算式だけでレンダリングを行なう不自由さから解放される。しかし、実際にレンダリングをすると、サーフェイスシェーダーの比重が大きいことに気付く。つまり、サーフェイスシェーダーは、それぞれの物体の材質などに依存するものであり、数多く存在することになる。よって、このサーフェイスシェーダーをさらに分類して整理する必要があると考えられる。

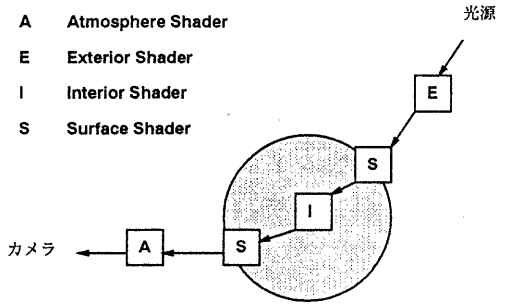

図 4: RenderMan に於けるレンダリングの過程

2.5 Block Shader

一方、shade tree を GUI (Graphical User Interface) を使って対話的に設計できるシステムを、Gregory. D. Abram と Turner Whitted が開発している [Abram 90]。このシステムでは、予め組み込まれているモジュール (レンダリングの計算に関するモジュール) を、ユーザーが組み合わせることで、あたかも回路を配線するように目的のシェーディングモデルを作成する。

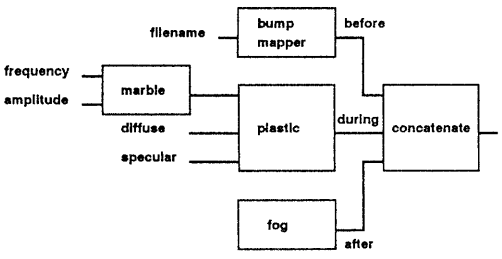

図 5: Building block shader

また、より低レベルのインターフェイスとして、シェーディング言語を開発していて、細かい設定を、直接このシェーディング言語でも開発できる。ただし、このシステムは shade tree の実装の実例であり、シェーディングモデルを作成するには、CG のレンダリングに関する専門知識をユーザーが持っていることが前提条件となっている。つまり、shade tree が残しているシェーディングモデルの本質的な設計の難しさは、そのまま引き継がれてしまっている。

この他にこのような GUI を採り入れたグラフィックシステムとして、Paul Haeberli の **ConMan**[Haeberli 88] などが挙げられる。

3 システムの概要

3.1 Shading Model の設計

本システムでは、シェーディングモデルをメッセージという概念で設計する。その様子を図 6に示す。

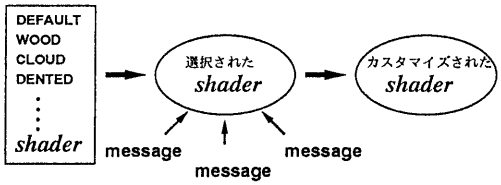

図 6: メッセージによるシェーダーの制御

図 6のように、予め定義されているシェーダー (つまり、shade tree の実体) の中から、ユーザーが作成するシェーディングモデルに近いシェーダーを選択し、それに目的に合わせてメッセージを木構造に組み合わせて送ることで、シェーダー自体をカスタマイズする。カスタマイズされたシェーダーは、任意の物体にバインドされる。物体の幾何データは外部から入れる。

例えば、本システムに予め用意されているデフォルトのシェーダーである "DEFAULT" は、Turner Whitted が [Whitted 80] で示した照明モデルを採用している。ユーザー自身は、予め用意されているシェーダーと全くかけはなれたシェーディングモデルを設計する場合を除いて、shade tree を作る必要はない。実際にシェーディングモデルを設計するに当たっては、それぞれのシェーダーやメッセージの検索や、その定義などに GUI を補助的に使用することで、作業の負荷の軽減をはかっている。本システムでは、この GUI をシェーディングに関するデータベースやメッセージなどの検索ツールとして、ユーザーが作業時に補助的に使用することを前提としている。

3.2 シェーディングモデルのデータベース化

さまざまなシェーディングモデルが開発され、その量が増えてくると、今度はそれらをいかにうまく再利用するかという問題が生じる。たまたま、あるシェーディングモデルが設計できたとしても、もしかしたら、それは別のシェーディングモデルを少し修正するだけで良かったかも知れない。また、シェーディングモデルの設計に

関するノウハウが蓄積する必要もある。本システムでは、図7に示しているように、シェーダーにオブジェクト指向モデルにおけるクラス階層を採り入れている。

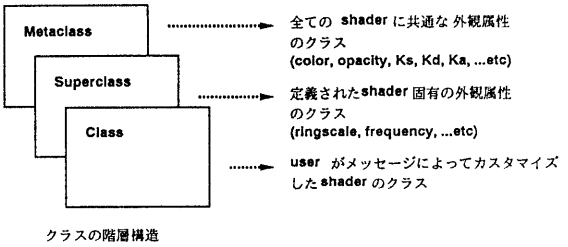

図 7: シェーダーのクラス階層

まず、メタクラスとして、全てのシェーダーに共通な外観属性をクラス定義する。共通な外観属性として、color、opacity、Ks、Kd、Kaなどの各係数を定義している。このメタクラスは全てのシェーダーが継承する。

また、スーパークラスとして、それぞれ定義されたシェーダーが固有に持っている外観属性をクラス定義する。例えば、"WOOD" というシェーダーでは、ringscaleやfrequencyといった属性を内部に持っている。

さらに、ユーザーがメッセージによってカスタマイズしたシェーダーをデータベース化する場合には、スーパークラスのサブクラスとして定義する。このカスタマイズされたシェーダーは、次からシェーダーとして再利用することが出来るようになる。実際にデータベース化されているクラスの例を図8、図9、図10に示す。

これらの図に示したように、クラス定義は上から、クラス名、継承するクラス、属性とその値、受け付けるメッセージとその実体へのポインタ (場所) となっている。

また、シェーダーのクラス定義で使われているメッセージを定義した例を図11、図12に示す。メッセージの定義には、受け付けるキーワード (つまり、値の評価を行なわずそのもの自体が値として用いられるもの) や、受け付けるメッセージ、さらにメッセージの実体へのポインタが書かれている。あるシェーダーにメッセージが送られてきた時の、メッセージの評価時にはこの定義が用いられる。

図11、図12に示したように、通常のオブジェクト指向モデルで用いられるメッセージの概念とは異なり、メッセージ自体にも、メッセージが送られてくる。このように、メッセージは単層構造ではなく木構造を形成する。

CLASS: OBJECT

```
(
 (attribute      OBJECT)
 (specialized    nil)
 (properties     (
                  (red          1.0)
                  (green        1.0)
                  (blue         1.0)
                  (Ks           0.5)
                  (Kd           0.5)
                  (Ka           0.5)
                  (Kr           0.0)
                  (Kt           0.0)
                  (refractance  1.0)
                  (roughness    0.5)
                  (opacity      0.0)
                  (shadow       0.0)
                  (ppf          nil)
                  ))
 (messages  (
             (keyword          nil)
             (RED              RED)
             (GREEN            GREEN)
             (BLUE             BLUE)
             (KS               KS)
             (KD               KD)
             (KA               KA)
             (KR               KR)
             (KT               KT)
             (REFRACTANCE      REFRACTANCE)
             (ROUGHNESS        ROUGHNESS)
             (OPACITY          OPACITY)
             (SHADOW           SHADOW)
             (PPF              PPF)
             (CHANGE-COLOR     CHANGE-COLOR)
             ))
)
```

図 8: メタクラスのクラス定義の例

CLASS: WOOD

```
(
 (attribute      WOOD)
 (specialized    OBJECT)
 (properties     (
                  (ringscale    0.02)
                  (lightlimit   0.8)
                  (darklimit    0.83)
                  (mixrange     0.0)
                  ))
 (messages       (
                  (keyword          nil)
                  (RINGSCALE        RINGSCALE)
                  (LIGHTLIMIT       LIGHTLIMIT)
                  (DARKLIMIT        DARKLIMIT)
                  (CHANGE-TEXTURE   CHANGE-TEXTURE)
                  (CHANGE-MATERIAL  CHANGE-MATERIAL)
                  (CHANGE-MIXRANGE  CHANGE-MIXRANGE)
                  ))
)
```

図 9: スーパークラス WOOD のクラス定義の例

CLASS: CLOUD

```
(
 (attribute        CLOUD)
 (specialized      DEFAULT)
 (properties   (
                (upper_value   1.0)
                (lower_value   0.0)
                (scale_x       40.0)
                (scale_y       40.0)
                (scale_z       40.0)
                ))
 (messages     (
                (keyword              nil)
                (UPPER_VALUE          UPPER_VALUE)
                (LOWER_VALUE          LOWER_VALUE)
                (CHANGE-SCALE         CHANGE-SCALE)
                (CHANGE-TEXTURE       CHANGE-TEXTURE)
                ))
)
```

図 10: スーパークラス CLOUD のクラス定義の例

MESSAGE: CHANGE-COLOR

```
(
 (message              CHANGE-COLOR)
 (acceptable
      (keyword         :REDDISH)
      (keyword         :GREENISH)
      (keyword         :BLUEISH)
      (keyword         :SKY-BLUE)
      (keyword         :BROWNISH)
      (keyword         :DARKEN)
      (keyword         :LIGHTEN)
      (RED             RED)
      (GREEN           GREEN)
      (BLUE            BLUE)
   )
 (process              CHANGE-COLOR)
)
```

図 11: メッセージ CHANGE-COLOR の定義の例

MESSAGE: CHANGE-TEXTURE

```
(
 (message              CHANGE-TEXTURE)
 (acceptable
      (keyword         nil)
      (TEXTURE-NAME    TEXTURE-NAME)
      (BLEND           BLEND)
   )
 (process              CHANGE-TEXTURE)
)
```

図 12: メッセージ CHANGE-TEXTURE の定義の例

4 システムの分散環境

本システムのサーバー側には、データベースマネージャーとレンダリングマネージャーが、クライアント側にはシステムマネージャーが用意されている。データベースマネージャーは、シェーダーのクラスのデータ管理やメッセージ木の評価 (シェーディングパラメータへの変換) を行ない、レンダリングマネージャーは、ユーザーが設計したシェーダーを利用して、レンダラーを生成 (コンパイル) する。一方、システムマネージャーは、ユーザーとサーバーとのインターフェイス部分を担う。

5 メッセージ木

5.1 メッセージ木の特徴

本システムでは、ユーザーがシェーディングモデルを作成する手段としてメッセージ木 (message tree) を用意している。つまり、シェーディングモデルをシェーディングのアルゴリズムの観点から作成するのではなく、より人間の感性に近い、CG の専門知識を特に持たないユーザーにも使い易いインターフェイスを提供している。

ここでは具体的にメッセージ木とは何かについて示す。本システムに於けるメッセージの主な特徴を挙げると次のようになる。

- シェーダー自身が内部に持っているシェーディングパラメータの変更ルールを内蔵した独立したプログラムである。
- メッセージ木自体はプログラミング言語ではなく、シェーディングパラメータの変更は各メッセージが独立して自主的に行なう。つまり、メッセージ木の評価の副作用として新しいシェーディングパラメータを計算する。
- 各メッセージは、プロセスとして並行に起動される。
- メッセージ木の評価中にルールが衝突した場合は、扱う属性値が数値であればその平均値を計算し、それ以外は先優先とする。
- メッセージの送り先 (親プロセス) によって別のルールを起動できる。
- メッセージの多重定義を許す。

メッセージ木を用いてシェーディングモデルを記述した例を図 13に示す。ここでは、"WOOD" というシェーダーに対してさまざまなメッセージを送っている。

```
(WOOD    (CHANGE-TEXTURE
             (TEXTURE-NAME      :FLOWER)
             (CHANGE-COLOR      :BROWNISH)
             (BLEND             :SUBTLE) )
         (CHANGE-COLOR
             (RED
                 (ADD           :A-LITTLE))) )
```

(((茶色がかった 花の 模様の) (少し 混ざった))
(赤を (少し 加えた)) 木)

図 13: Message tree

このメッセージ木は自然言語に於ける修飾語 (形容詞) との類似性がある。つまり、ある物の様子を表現する場合にその物に修飾語が付けられるように、あるシェーダーをどのように変更するかを表現したものがメッセージ木となる。例えば図 13では、「茶色がかった花の模様の、少し混ざった、赤を少し加えた、木」を表現している。それぞれの修飾語の依存関係を括弧で示す。

メッセージ木は、ユーザーがシェーディングモデルを設計する際に、シェーディング言語のようなプログラムによるアプローチではなく、また、シェーディングのアルゴリズムの観点からシェーディングモデルを定義するものでもない。メッセージ木では、それぞれのメッセージの送り先 (つまり、そのメッセージを起動した親プロセス) によって、メッセージの内部の別のルールを起動することが出来る。例えば図 13に於いて、"CHANGE-COLOR" というメッセージが 2 回使われているが、それぞれメッセージの送り先が異なるので、違ったルールが起動される。

このように、メッセージの送り先によって別のルールが起動でき、また多重定義が出来ることで、メッセージの種類を比較的少なく押えることが出来る。それによって、ユーザーのメッセージの検索の手間を軽減でき、シェーディングモデルの設計に専念することが出来る。

また、シェーディングモデルの表現が、数値だけのパラメータ表現ではないので、データベース化 (ライブラリー化) したときにデータの扱いが容易になる。

6 レンダリング

本システムを用いて実際にレンダリングを行なった例を示す。図 14は、図 16によって定義したシェーディングモデルをレンダリングしたものである。 METAL は、メタクラスである OBJECT の属性を継承しつつ、固有な属性も持っている。図 16は、その中の色を変えたものである。また、図 15は、図 17によって定義したシェーディングモデルをレンダリングしたものである。

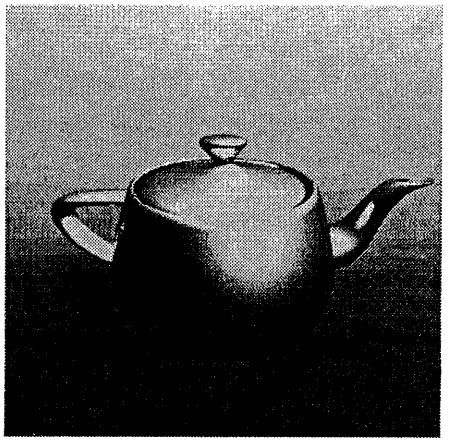

図 14: 金属の質感を持ったティーポット

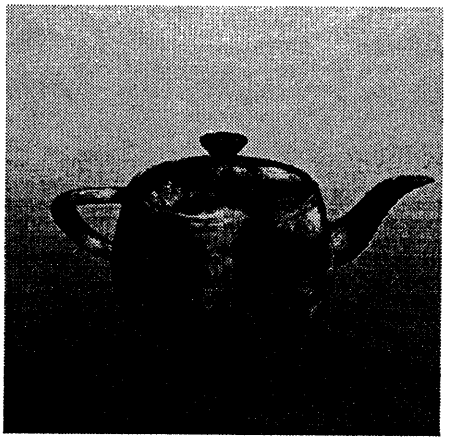

図 15: ソリッドテクスチャ + 2 次元マッピングによるティーポット

```
(METAL
    (CHANGE-COLOR       :SKY-BLUE)
  )
```

図 16: 図 14 の画像生成に使用したメッセージ木

```
(CLOUD
    (CHANGE-COLOR           :REDDISH)
    (CHANGE-TEXTURE
        (TEXTURE-NAME       :YELLOW-FLOWER)
        (BLEND              :LARGELY)
        (SOFTEN             :A-LITTLE)
        (TEXTURE-PATTERN
            (GRANULATE      :A-LITTLE)
        )
    )
)
```

図 17: 図 15 の画像生成に使用したメッセージ木

7 結論

7.1 結論

本研究では、CG の専門知識を特に持たないユーザーが、シェーディングモデルの設計が出来るようなシェーディングモデル作成支援システムを提案した。本システムにより、メッセージ木を用いた簡略な記述だけでシェーディングモデルが表現できた。

本システムでは、shade tree の概念も採り入れ、CG の専門知識を特に持たないユーザーが、シェーディングのアルゴリズムを組み立てずにシェーディングモデルを設計できるようにする為に、オブジェクト指向モデルの概念に基づいて、メッセージ木という従来のものと異なったインターフェイスを提供している。これによって、シェーディングモデルを、より人間の感性に近い形で表現することが可能になった。

ユーザーは、予めシステムに用意されているシェーダーとメッセージを組み合わせることで、容易にシェーディングモデルが表現できるようになったが、システムに用意されていないシェーダーを利用する場合や、より複雑なシェーディングモデルを設計する為には、実際に、シェーディングのプログラム (アルゴリズム) をユーザー自身が組み立てる必要がある。よって、メッセージ木を用いて表現したシェーディングモデルから、シェーディングのプログラム (アルゴリズム) への変換の機能を備えることが望ましいと考えられる。

謝辞

本研究を行なうに当たり、慶應義塾大学理工学部電気工学科 小澤 慎治 教授、及び大野 義夫 助教授には、研究の場を与えて下さり、また適切な御指導をして頂きました。この場を借りて、心から感謝の意を表したいと思います。

参考文献

[Whitted 80] Turner Whitted, "An Improved Illumination Model for Shaded Display," *Communications of the ACM* 23 pp.343-349 (1980).

[Cook 84] Robert L. Cook, "Shade Trees," *Computer Graphics* 18(3) pp.223-231 (July 1984).

[Perlin 85] Ken Perlin, "An Image Synthesizer," *Computer Graphics* 19(3) pp.287-296 (1985).

[Haeberli 88] Paul Haeberli, "ConMan: A Visual Programming Language for Interactive Graphics," *Computer Graphics* 22(4) pp.103-111 (July 1987).

[Pixar 88] Pixar Corporation, *The RenderMan Interface*, Version 3.0, Pixar Corporation, San Rafael, CA, May 1988

[Upstill 89] Steve Upstill, *The RenderMan Companion*, Addison-Wesley, (Reading, Massachusetts), 1989.

[Hanrahan 90] Pat Hanrahan and Jim Lawson, "A Language for Shading and Lighting Calculations," *Computer Graphics* 24(4) pp.289-298 (1990).

[Abram 90] Gregory D. Abram and Turner Whitted, "Building Block Shaders," *Computer Graphics* 24(4) pp.283-288 (1990).