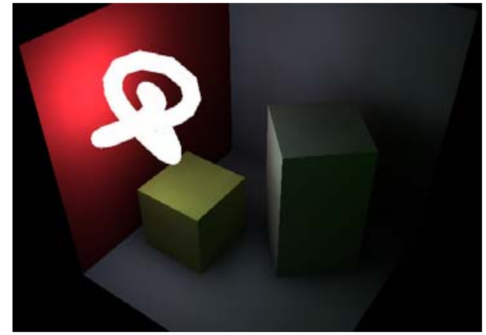
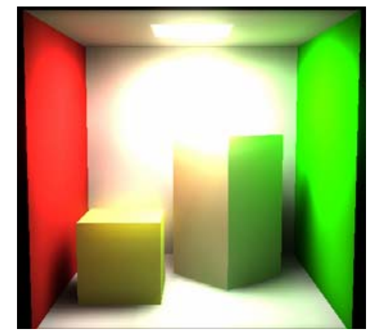
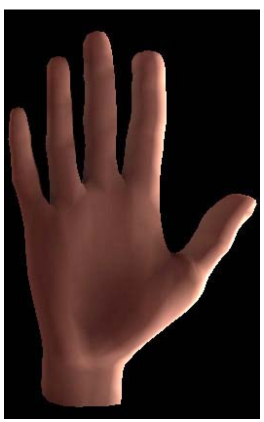


ゲームプログラマーのための初級PRT





(株)トライエース 研究開発部
五反田 義治

CEDEC 2006
CESA DEVELOPERS CONFERENCE 2006

tri-Ace Inc.
Research and Development Department

PRT

- Global Illuminationを現実的なパフォーマンスでリアルタイムに処理することができる手法の一つ
  - Peter-Pike Sloan et al. “Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments.” SIGGRAPH 2002

Outline

1. PRTとは?
2. **基底変換とデータ圧縮**
3. **レンダリング**
4. PCA

Outline

1. PRTとは?
    1. PRT概要
    2. PRTの種類
2. 基底変換とデータ圧縮
3. レンダリング
4. PCA

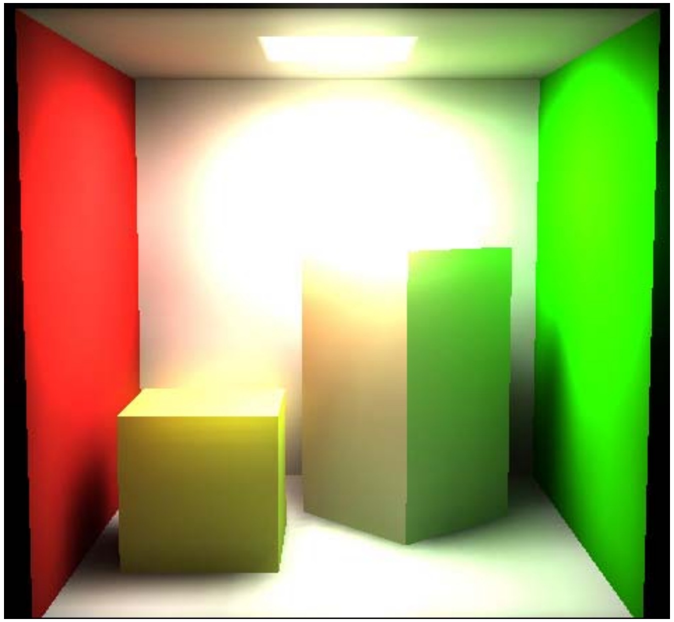
このセッションの目的

- 現在PRTはそのアルゴリズムが多岐に渡っている
  - 論文に数式が多く、一見難解
    - しかし、PRTの考え方はシンプル(PS2でもトライできる)
  - PRTの基礎的な部分を理解できれば
    - 派生系の論文を理解することができるはず
  - PS2での実装例は以下をご覧ください
    - **Yoshiharu Gotanda, Tatsuya Shoji "Practical Implementation of SH Lighting and HDR Rendering on PlayStation 2" GDC 2005**
      - http://research.tri-ace.com/

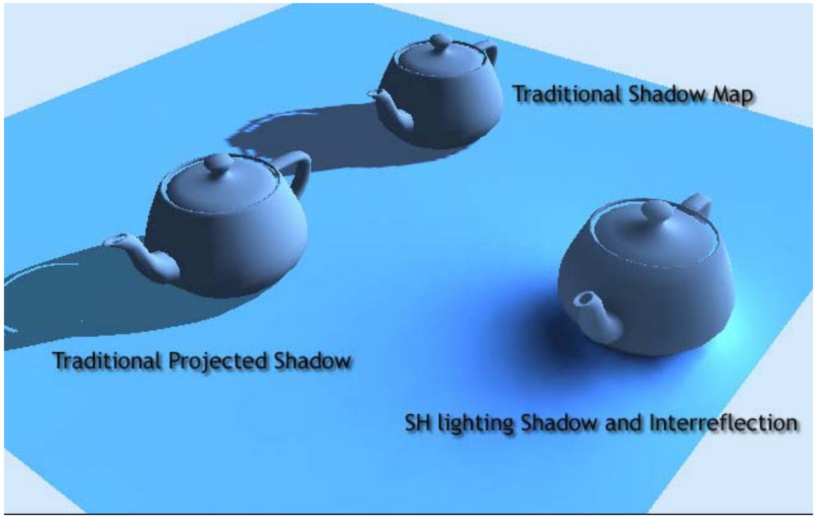
PRTでできること

- 限定的なReal-time Global illumination
  - Diffuse的なエフェクト
    - 影(Occlusion)
    - 相互反射(Interreflection)
    - 表面下散乱(Subsurface scattering)
    - 任意のBRDF
    - イメージベーストライティング(IBL)
  - 限定的なSpecularエフェクト

PRT例(1)

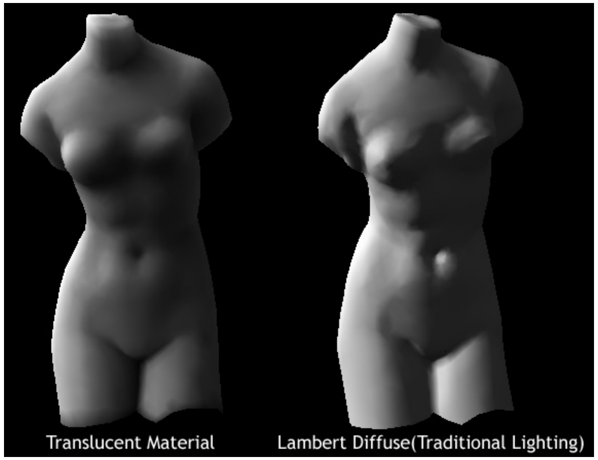


Global Illumination



Soft Shadow

PRT例(2)



Translucent Materials

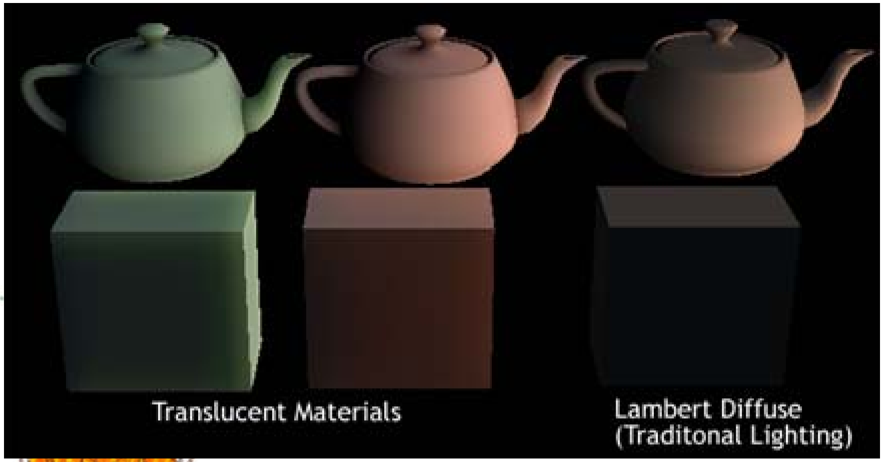


Image Based Lighting

PRT例(3) – 比較

	係数の数	SHデータサイズ (bytes)	VU1での命令数	速度比	データサイズ比（テクスチャなし）
従来のライティング (4dir+1amb)	0	0	10(15)	1.00	1.00
SH：2bands – 1ch	4	8	6(13)	1.05	1.37
SH：4bands – 1ch	16	32	21(28)	2.07	2.83
SH：2bands – 3chs	12	24	9(16)	1.57	2.00

注)スクリーンショットは実際の効果のサンプルで、測定に利用したデータではありません

（）セカンダリライトシェーダを含む

セカンダリシェーダとは最終色の計算とクランピングを行なう処理を指す

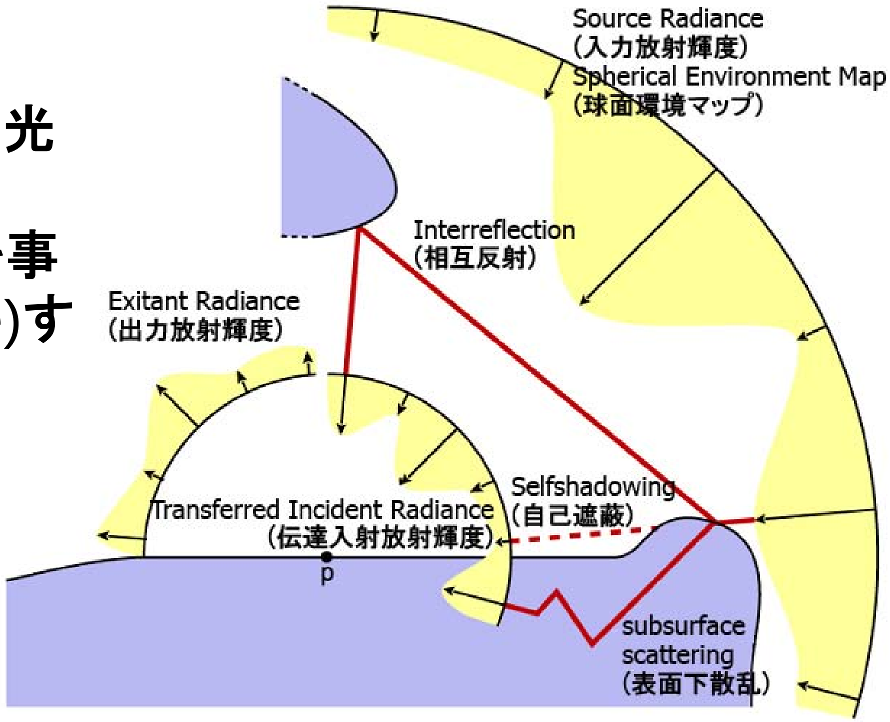
GDC2005のスライドから抜粋

PRTの制限

- 再現に制限のあるエフェクト
  - オブジェクトの変形に制限がある
    - 手法に依存
  - ライト環境やカメラのアニメーションに負荷が高い手法がある
  - Specularライティングに関してハードルが高い
    - 手法によって様々

PRTとは(1)

- あるサーフェースの1点において
  - すべての方向から来た光に対して反射する光(Radiance Transfer)を事前に計算(Precompute)すること



Courtesy of Peter-Pike Sloan

PRTとは(2)

- 要はライティング計算をオフラインで先に計算しておくだけ
  - PRTはテーブルでライティングするようなイメージ
    - テーブル=データ
  - しかし、データ量と演算量が爆発!
    - 例えばper-vertexで考えると
      - 10,000頂点に10,000方向のデータを計算したら、100,000,000要素のデータがこのオブジェクトに必要

PRTとは(3)

- まずはデータ量が問題
  - Diffuseだけだとしても
    - RGB 16bitずつで10,000頂点で10,000方向なら572MB!
  - データ量を減らすために圧縮する
    - 例えば、LZやHuffmanで圧縮すると
      - 展開コストが上乗せされてしまう
      - あんまり圧縮が効かない

PRTとは(4)

- 演算量も問題
  - IBL(Image Based Lighting)の場合、計算した方向の数すべてにライトがあるようなもの
    - 10,000方向あれば10,000個分のライト演算
    - ただし、結果は事前に計算されている
      - 計算処理はシンプル

```
for(int i = 0; i < 10000; i++) {
   r+= light[i].r*prt[i].r;   g+=light[i].g*prt[i].g;   b+=light[i].b*prt[i].b;
}
```

PRTとは(5)

- 演算量とデータ量を減らすためにデータ圧縮を行なう
  - 大幅なデータ量削減が必要
    - 非可逆(lossy)圧縮を行なう
  - 演算量も減らしたいので0の要素を増やしたい
    - 0なら掛ける必要が無い

PRTまとめ

- PRTとは
  - オフラインでライティングに必要な計算を先にしておく
    - オフラインでどこまで計算しておくかは手法に依存する
    - オフラインでの計算により最終的なクオリティも変わってくる
  - データを圧縮して持っておき、リアルタイムで展開しながらレンダリングする

圧縮方法

- 基底変換を行なう
  - これが今日のセッションの肝
  - その前に…現在のPRTはどのような感じか

Outline

1. PRT とは?
   1. PRT概要
   2. PRTの種類
2. 基底変換とデータ圧縮
3. レンダリング
4. PCA

SH

- Spherical Harmonics(球面調和関数)を基底とした基底変換
  - Peter-Pike Sloan et al. “Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments.” SIGGRAPH 2002
  - 高周波データには大量の係数が必要
  - SHの回転を行なうことが出来る(アニメーションの可能性)
  - 直交関数
  - レンダリングは線形

Image from the paper

Wavelet

- 基底にWavelet関数を利用したデータ圧縮
  - Ren Ng et al. “All-Frequency Shadows Using Non-Linear Wavelet Lighting Approximation” SIGGRAPH 2003
  - Ren Ng et al. “Triple Product Wavelet Integrals for All-Frquency Relighting.” SIGGRAPH 2004
  - 高周波データにも適応
  - 回転はできない
    - ただし「Rui Wang et al. “Efficient Wavelet Rotation for Environment Map Rendering” EGSR 2006」でWavelet回転について触れている)
  - 直交関数
  - レンダリングは非線形
  - 効率は母関数に依存する

Images from the papers

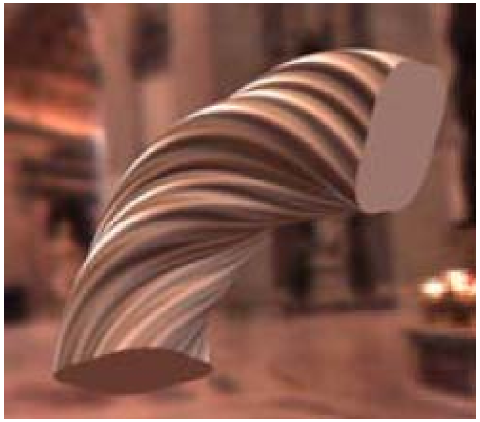
SH(or Wavelet) + CPCA

- Clustered Principal Component Analysis(クラスタ化された主成分分析)を利用したデータ圧縮
  - SH(or Wavelet)基底に投影されたPRTデータをさらにCPCAで圧縮
  - (SH+CPCA) Peter-Pike Sloan et al. “Clustered Principal Components for Precomputed Radiance Transfer” SIGGRAPH 2003
  - (BRDF factorization + Wavelet + CPCA) Peter-Pike Sloan et al. “All-Frequency Precomputed Radiance Transfer for Glossy Objects” EGSR 2004
  - 高周波データにも適応
  - PCA部分では回転はできない
  - 直交している
  - レンダリングは線形(SH)、非線形(Wavelet, CPCA)

Image from the paper

LDPRT

- ZH(Zonal Harmonics)を利用したデータ圧縮
  - Peter-Pike Sloan et al. “Local, Deformable Precomputed Radiance Transfer” SIGGRAPH 2005
  - 高周波データには大量の係数が必要
  - ZHの回転はSHに比べて負荷が低い
    - アニメーションが簡単
  - 直交関数
    - というよりかなり近似
  - レンダリングは線形



Images from the paper

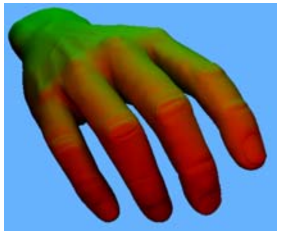
SRBF + CTA

- SRBF(Spherical Radial Basis Functions)に投影したPRTデータをCTA(Clustered Tensor Approximation)を利用して圧縮
  - Yu-Ting Tsai et al. “All-Frequency Precomputed Radiance Transfer using Spherical Radial Basis Functions and Clustered Tensor Approximation” SIGGRAPH 2006
  - Diffuse PRTにはCPCAを利用している
  - 高周波データにも適応
  - 回転可能
  - 直交関数ではない
  - 圧縮率はかなり良い

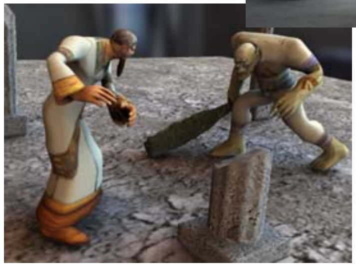
Image from the paper

その他

- T. Annen et al. “Spherical Harmonic Gradients for Mid-Range Illumination” EGSR 2004
  - SHの勾配(微分)を持っておくことにより近距離のライティングを可能にする
- J. Kautz et al. “Hemispherical Rasterization for Self-Shadowing of Dynamic Objects” EGSR 2004
  - リアルタイムにマスクを生成し、そこからSHを計算し、シャドウを生成する
- Paul Green et al. “View-Dependent Precomputed Light Transport Using Nonlinear Gaussian Function Approximations” i3D 2006
  - ガウス関数を利用した非線形近似
- Kun Zhou et al. “Precomputed shadow fields for dynamic scenes” SIGGRAPH 2005
  - オクルージョンフィールドとライトフィールドをPRTとして持ち、アニメーション可能なシャドウを実現する
- Zhong Ren et al. “Real-time Soft Shadows in Dynamic Scenes using Spherical Harmonic Exponentiation” SIGGRAPH 2006
  - リアルタイムにSHを計算し、IBLに対してリアルタイムのソフトシャドウを実現する。TripleProduct高速化のためにSHを対数化している
- まだまだ、たくさんのPRT関連の資料はあります







Images from the papers

Outline

1. PRTとは?
2. **基底変換とデータ圧縮**
   1. **基底変換**
   2. 最小二乗法
   3. 直交変換
3. レンダリング
4. PCA

どうやって圧縮するのか

- 大量のPRTデータのうち、いらないデータを省く
  - どのように必要の無いデータを見つけるのか?
  - どのようにデータを省くのか?
  - どのようにレンダリングを行なうのか?

基底変換

- 基底変換を利用
  - すると…
    - いらないデータが見えてくる
    - データを省くための特性も見えてくる
    - 線形であればレンダリングも基本的に内積
      - データが小さくなる
      - 高速化にもつながる

基底変換とは?

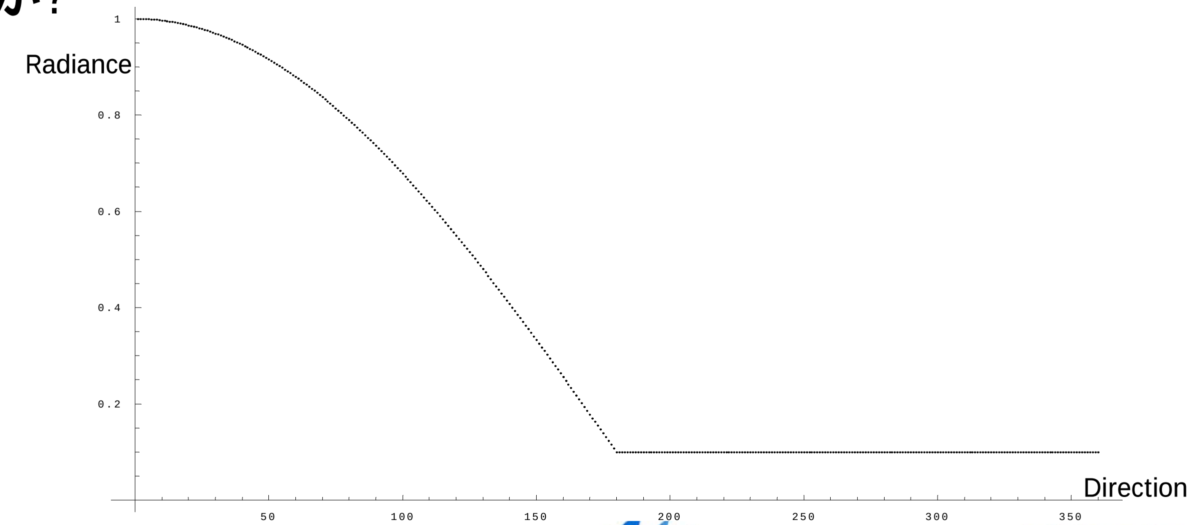
- ある関数を、違う関数(基底関数)での積和(線形結合)で表現すること
  - 例えばある関数 $g(x)$ が定義されているときに

$$\underbrace{g(x)}_{\text{表現したい関数またはデータ列}} \approx k_0 \cdot f_0(x) + k_1 \cdot f_1(x) \ldots + \underbrace{k_n}_{\text{係数}} \cdot \underbrace{f_n(x)}_{\text{基底関数}}$$

  - このように別の関数で表現する

より詳しく(1)

- PRTではこの関数 $f(x)$ がデータ列として離散的に表現されている
  - このままではデータの数だけのデータ量が必要になってしまう
  - なんとかデータ量を減らすために簡単な形であらわせないか?



より詳しく(2)

- とりあえず、一次関数 $f(x) = ax + b$ で表してみる
  - うまくいけば、パラメータはaとbの2個
    - しかし、どのようにaとbを求める?

Outline

1. PRT とは?
2. 基底変換とデータ圧縮
    1. 基底変換
    2. 最小二乗法
    3. 直交変換
3. レンダリング
4. PCA

最小二乗法

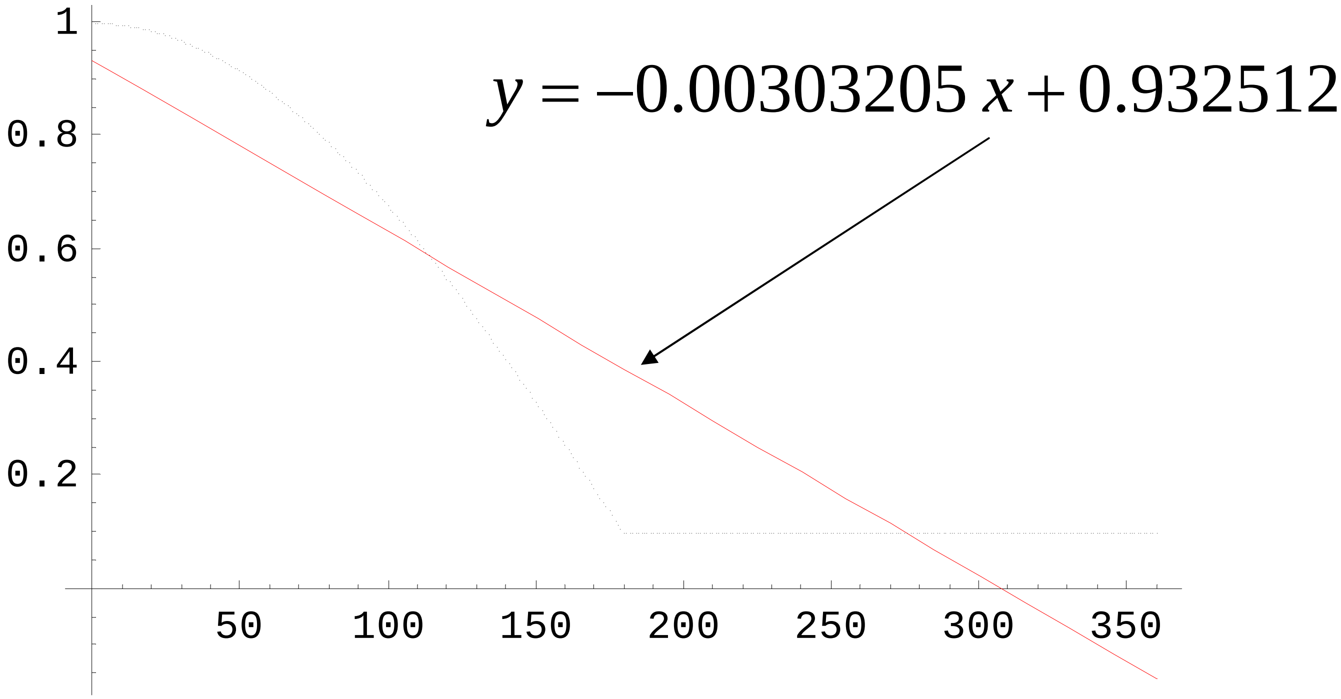
- 最小二乗法を利用してみる
  - フィッティングを行なうデータ列gと関数f(x)のすべての値の差の二乗和を最小にする

$$J = \frac{1}{2}\sum_{k=1}^{N}(g_k - f(k))^2 \to \min$$

```
// 例えば一次関数f(x)=ax+bにフィッティングするなら
float J = 0.0f;
for(int k = 1; k <= N; k++)
   J += powf(g[k - 1] – (a*k + b), 2.0f);
// このときのJが最小になるaとbを見つける
```

最小二乗法

- このデータでは?



計算法

$$J = \frac{1}{2}\sum_{k=1}^{N}(g_k - f(k))^2 \rightarrow \min$$

- この関数 Jが最小値を取るこということは、その値は極小値であるはず
  - 極小値である場所は勾配(微分係数)が0になる
    - 関数 Jを微分した式の値が0になる値が最小値であるはず
- では先ほどの一次関数 ax+bでは?

偏微分

- 多変数の微分
  - 複数の変数がある式に対して、それぞれの変数ごとに微分を行なうことを偏微分という
    - 計算自体は、偏微分を行なう変数以外を定数とみなして微分するだけ

$f(x, y)$ という関数について

$\dfrac{\partial f}{\partial x} =$ xについて偏微分するときは、yを定数とみなす

$\dfrac{\partial f}{\partial y} =$ yについて偏微分するときは、xを定数とみなす

一次関数の最小二乗法(1)

- 一次関数ax+bでは、aとbの値を求めたい
  - この場合Jはaとbの関数になる
    - aとbにおける偏導関数が0になればよい

$$J=\frac{1}{2}\sum_{k=1}^{N}(g_k-(ax_k+b))^2$$

$$=\frac{1}{2}\sum_{k=1}^{N}(x_k^2a^2+2x_kba-2g_kx_ka-2g_kb+b^2+g_k^2)$$

$$\frac{\partial J}{\partial a}=a\sum_{k=1}^{N}x_k^2+b\sum_{k=1}^{N}x_k-\sum_{k=1}^{N}x_kg_k$$

$$\frac{\partial J}{\partial b}=a\sum_{k=1}^{N}x_k+b\sum_{k=1}^{N}1-\sum_{k=1}^{N}g_k$$

一次関数の最小二乗法(2)

- 偏導関数 $\dfrac{\partial J}{\partial a}, \dfrac{\partial J}{\partial b}$ が0になればいいということは…
  - 以下の連立一次方程式(正規方程式)の解を求めればいいことになる

$$
\begin{pmatrix} \sum_{k=1}^{N} x_k^2 & \sum_{k=1}^{N} x_k \\ \sum_{k=1}^{N} x_k & \sum_{k=1}^{N} 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{N} x_k g_k \\ \sum_{k=1}^{N} g_k \end{pmatrix}
$$

一次関数の最小二乗法(3)

- プログラムで書くと…

```
float im[2][2], m[2][2] = {0.0f, 0.0f, 0.0f, 0.0f}, bx = 0.0f, by = 0.0f;
// 各変数の和を求める
for(int x = 1; x <= N; x++) {
  bx += x * data[x - 1];  by += data[x - 1];
}
m[0][0] = N * (1.0f + N) * (1.0f + 2.0f * N) / 6.0f;
m[1][0] = m[0][1] = 0.5f * N * (1.0f + N);
m[1][1] = N;
// 逆行列を求めて、解a,bを計算
float det = 1.0f / (m[0][0] * m[1][1] – m[0][1] * m[1][0]);
im[0][0] = m[1][1] * det; im[0][1] = -m[0][1] * det;
im[1][0] = -m[1][0] * det; im[1][1] = m[0][0] * det;
float a = im[0][0] * bx + im[0][1] * by;
float b = im[1][0] * bx + im[1][1] * by;
```

$$\sum_{x}^{N} x^2 = \frac{N(1+N)(1+2N)}{6}$$

$$\sum_{x}^{N} x = \frac{N(1+N)}{2}$$

一次関数の最小二乗法(4)

- このように一次関数でデータを表すことは、簡単で計算も単純である
  - 当然リニアに近いデータでなければ誤差も大きい
- パラメータが増えたとしても他の関数で表せないか?
  - 例えば二次関数 $f(x) = ax^2 + bx + c$

二次関数の最小二乗法

- 今度はa,b,cの3つの値を求めたい
  - 同じように、a,b,cにおける偏導関数が0になればよい

$$J = \frac{1}{2}\sum_{k=1}^{N}(g_k - (ax_k^2 + bx_k + c))^2$$

$$\frac{\partial J}{\partial a} = a\sum_{k=1}^{N} x_k^4 + b\sum_{k=1}^{N} x_k^3 + c\sum_{k=1}^{N} x_k^2 g_k - \sum_{k=1}^{N} x_k^2 g_k$$

$$\frac{\partial J}{\partial b} = a\sum_{k=1}^{N} x_k^3 + b\sum_{k=1}^{N} x_k^2 + c\sum_{k=1}^{N} x_k - \sum_{k=1}^{N} x_k g_k$$

$$\frac{\partial J}{\partial c} = a\sum_{k=1}^{N} x_k^2 + b\sum_{k=1}^{N} x_k - c\sum_{k=1}^{N} 1 - \sum_{k=1}^{N} g_k$$

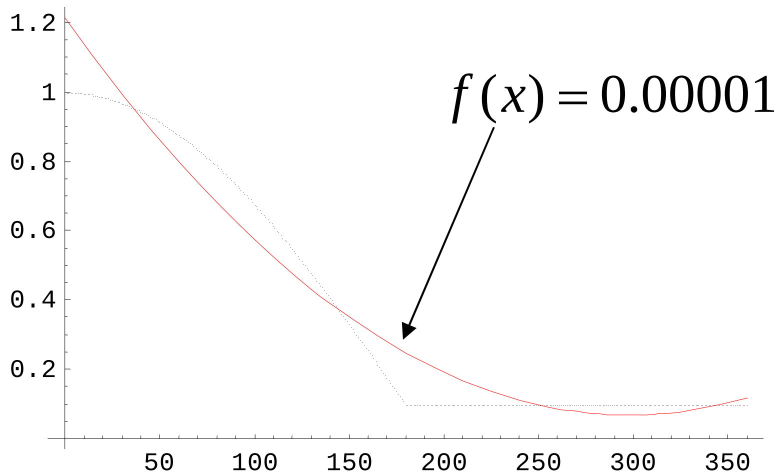
n次関数の最小二乗法

- 結局3x3の正規方程式ができる
  - 3次関数なら4x4…n-1次関数ならn×n
  - 結局、n次関数に対する最小二乗法の一般解は
    - 関数の係数を $c_0, c_1, \cdots, c_n$ とすると

$$
\begin{pmatrix}
\sum_{k=1}^{N} x_k^{2n} & \sum_{k=1}^{N} x_k^{2n-1} & \cdots & \sum_{k=1}^{N} x_k^{n} \\
\sum_{k=1}^{N} x_k^{2n-1} & \sum_{k=1}^{N} x_k^{2n-2} & \cdots & \sum_{k=1}^{N} x_k^{n-1} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{k=1}^{N} x_k^{n} & \sum_{k=1}^{N} x_k^{n-1} & \cdots & \sum_{k=1}^{N} 1
\end{pmatrix}
\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix}
=
\begin{pmatrix}
\sum_{k=1}^{N} x_k^{n} g_k \\
\sum_{k=1}^{N} x_k^{n-1} g_k \\
\vdots \\
\sum_{k=1}^{N} g_k
\end{pmatrix}
$$

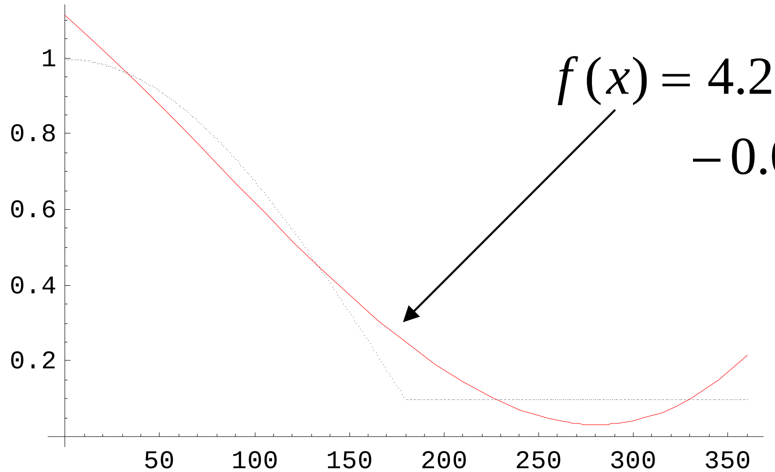
となる

フィッティング例



$$f(x) = 0.000012928x^2 - 0.007699x + 1.2141$$

高次の係数の値は小さい!



$$f(x) = 4.24284 \times 10^{-8} x^3 - 0.00001005x^2 - 0.00437685x + 1.11345$$

一般化された最小二乗法(1)

- フィッティングを行なう関数をn次関数から
  - 任意の関数の線形結合に一般化すると…

$$g_k \approx \sum_{t=1}^{n} c_t f_t(x_k)$$

n個の任意の関数の線形結合

$$J = \frac{1}{2}\sum_{k=1}^{N}\left(g_k - \sum_{t=1}^{n} c_t f_t(x_k)\right)^2 \to \min$$

この関数の値を最小化する
n個ある係数cを求めること
になる

一般化された最小二乗法(2)

- 二次関数を一般化された式と比べると

$$g_k \approx a x_k^2 + b x_k + c \cdot 1$$

それぞれが対応している

$$g_k \approx c_0 f_0(x_k) + c_1 f_1(x_k) + c_2 f_2(x_k)$$

$$f_0(x) = x^2,\ f_1(x) = x,\ f_2(x) = 1$$

一般化された最小二乗法(3)

- 同じように偏導関数が0になればいいのだから

$$\frac{\partial J}{\partial c_1} = 0, \Lambda \; \frac{\partial J}{\partial c_n} = 0$$

- 各係数cにおける偏導関数は以下のようになる

$$\frac{\partial J}{\partial c_i} = \sum_{k=1}^{N} \left(g_k - \sum_{t=1}^{n} c_t f_t(x_k) \right) \left(- f_i(x_k) \right)$$

$$= \sum_{t=1}^{n} \left(\sum_{k=1}^{N} f_t(x_k) f_i(x_k) \right) c_k - \sum_{k=1}^{N} f_i(x_k) g_k$$

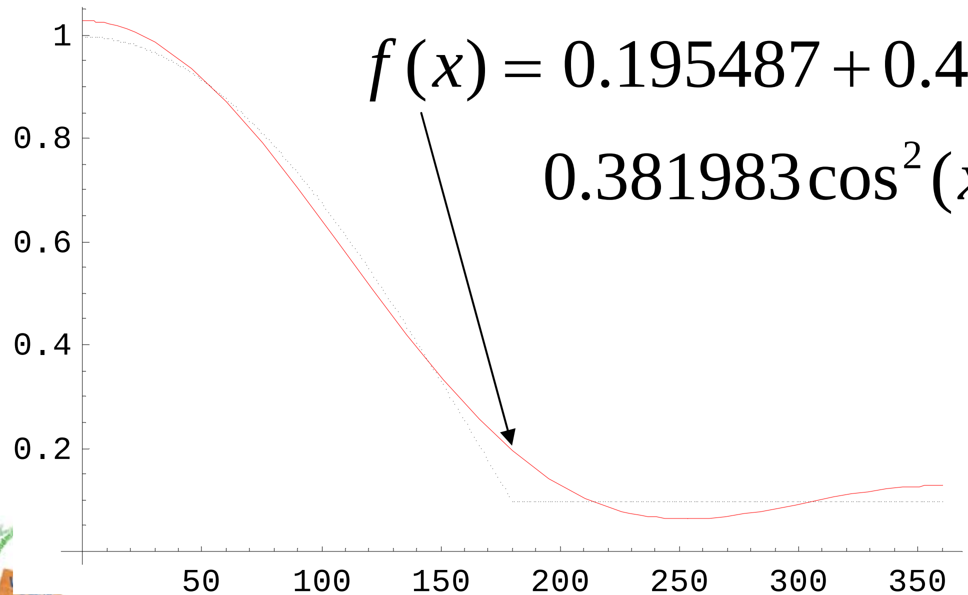
一般化された最小二乗法(4)

- 正規方程式は以下のようになる

$$\begin{pmatrix} \sum_{k=1}^{N} f_1(x_k)^2 & \sum_{k=1}^{N} f_1(x_k) f_2(x_k) & \cdots & \sum_{k=1}^{N} f_1(x_k) f_n(x_k) \\ \sum_{k=1}^{N} f_2(x_k) f_1(x_k) & \sum_{k=1}^{N} f_2(x_k)^2 & \cdots & \sum_{k=1}^{N} f_2(x_k) f_n(x_k) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{N} f_n(x_k) f_1(x_k) & \sum_{k=1}^{N} f_n(x_k) f_2(x_k) & \cdots & \sum_{k=1}^{N} f_n(x_k)^2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{N} f_1(x_k) g_k \\ \sum_{k=1}^{N} f_2(x_k) g_k \\ \vdots \\ \sum_{k=1}^{N} f_n(x_k) g_k \end{pmatrix}$$

フィッティング例

- 例えば$g_k \approx \sum_{t=1}^{n} c_t \cos^t(k)$でフィッティングしてみる



$$f(x) = 0.195487 + 0.449695\cos(x) + 0.381983\cos^2(x) + \underline{0.000610426}\cos^3(x)$$

やはり、高次の係数の値は小さい

圧縮の手順

- 最小二乗法を用いると、データを任意の線形結合された関数として表すことができる
  - 基底関数の選び方と数が誤差を決める
    - 一般的に適切な関数を用意すれば、データと同じ数だけの係数(次数)を用意すると、どのようなデータに対しても(数学的には)誤差は0になる
  - 数式上で誤差が0になるようにして、そのうち重要でない係数を0にすることにより圧縮が可能になる
    - MP3やJpegなどでは単純に0にするのではなく、重要でない係数の量子化ビット数を減らすことにより圧縮している
    - または最初から計算する基底の数を減らすこともできる

最小二乗法の問題点

- 係数を求めるということは、結局逆行列を求めるという作業になる
  - データが多いと巨大な行列の逆行列を求めることになる
    - 大きな行列を解析的に求めることは、現実的ではない
  - 数値計算手法を利用する
    - Gauss-Jordan法,LU分解,SVD…

逆行列

- 逆行列を求める処理はとにかく重い
  - 解析的には $O(N^3 N!)$
  - 3乗の上に階乗であっというまに爆発!
- 他の数値計算手法でも重い
  - 計算量はほぼ $O(N^3)$
- 何か他の手法はないか?

Outline

1. PRTとは?
2. **基底変換とデータ圧縮**
   1. 基底変換
   2. 最小二乗法
   3. **直交変換**
3. レンダリング
4. PCA

直交変換(1)

- よく正規方程式を眺めてみる

$$
\begin{pmatrix}
\boxed{\sum_{k=1}^{N} f_1(x_k)^2} & \sum_{k=1}^{N} f_1(x_k)f_2(x_k) & \cdots & \sum_{k=1}^{N} f_1(x_k)f_n(x_k) \\
\sum_{k=1}^{N} f_2(x_k)f_1(x_k) & \boxed{\sum_{k=1}^{N} f_2(x_k)^2} & \cdots & \sum_{k=1}^{N} f_2(x_k)f_n(x_k) \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{k=1}^{N} f_n(x_k)f_1(x_k) & \sum_{k=1}^{N} f_n(x_k)f_2(x_k) & \cdots & \boxed{\sum_{k=1}^{N} f_n(x_k)^2}
\end{pmatrix}
\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}
=
\begin{pmatrix}
\sum_{k=1}^{N} f_1(x_k)g_k \\
\sum_{k=1}^{N} f_2(x_k)g_k \\
\vdots \\
\sum_{k=1}^{N} f_n(x_k)g_k
\end{pmatrix}
$$

- 対角成分だけは他の部分と違って二乗になっている
- ほかの部分は必ず違う関数との積

直交変換(2)

- もし非対角成分を0にできるなら

$$
\begin{pmatrix}
\sum_{k=1}^{N} f_1(x_k)^2 & 0 & \cdots & 0 \\
0 & \sum_{k=1}^{N} f_2(x_k)^2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \sum_{k=1}^{N} f_n(x_k)^2
\end{pmatrix}
\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}
=
\begin{pmatrix}
\sum_{k=1}^{N} f_1(x_k) g_k \\
\sum_{k=1}^{N} f_2(x_k) g_k \\
\vdots \\
\sum_{k=1}^{N} f_n(x_k) g_k
\end{pmatrix}
$$

逆行列を求める必要が無い!

直交変換(3)

- 非対角成分を0にするということは…

$$\sum_{k=1}^{N} f_i(x_k) f_j(x_k) = 0 (i \neq j)$$

であればよい

- このような関数を直交関数系という
  - 一般的には以下のときに{ ϕ (x)}は[a,b]上の直交関数系となる

$$\int_a^b \phi_i(x)\phi_j(x)dx = 0 \ (i \neq j)$$

直交変換(4)

- 直交関数系で係数を求めるには
  - 逆行列を計算する必要は無く、以下の計算のみで各係数が求まる

$$c_i = \frac{\sum_{k=1}^{N} f_i(x_k) g_k}{\sum_{k=1}^{N} f(x_k)^2}$$

直交変換(5)

- 直交関数系を利用すると
  - 少ない計算量で基底変換を行なうことが出来る
- 一般的な直交関数
  - フーリエ(Fourier)級数
  - ルジャンドル(Legendre)多項式
  - チェビチェフ(Chebyshev)多項式
  - エルミート(Hermite)多項式
  - ラゲール(Laguerre)多項式…など

直交関数例

- SH LightingはPRTデータを球面調和関数に基底変換を行なう
  - 球面調和関数は正規直交関数

$$Y_l^m(\theta,\phi) \equiv \sqrt{\frac{2l+1(l-m)!}{4\pi(l+m)!}} P_l^m(\cos\theta) e^{im\phi}$$

$$m = -l, -(l-1), \ldots 0, \ldots (l-1), l$$

$P_l^m(z)$ はルジャンドル陪関数

基底変換まとめ

- 基底変換を利用すると
  - データ列や任意の関数をいくつかの基底と係数で近似できる
    - 非可逆圧縮が可能になる
    - 巨大なデータをドラスティックに圧縮できる
  - 直交関数系を利用すると高速に基底変換が可能
    - 直交関数のほうが近似精度が優れているとは限らない

Outline

1. PRTとは?
2. 基底変換とデータ圧縮
3. レンダリング
4. PCA

レンダリング(1)

- 圧縮されたPRTデータをどのようにレンダリング(シェーディング)を行なうのか?
  - そもそもレンダリングとは?

レンダリング(2)

- **レンダリング方程式(BRDF版)**

$$\underbrace{L_o(x,\boldsymbol{\omega})}_{\substack{\text{出射される光}\\\text{(放射輝度)}}} = \underbrace{L_e(x,\boldsymbol{\omega})}_{\text{自己放射}} + \int_\Omega \underbrace{f_r(x,\boldsymbol{\omega}',\boldsymbol{\omega})}_{\text{BRDF}} \underbrace{L_i(x,\boldsymbol{\omega}')}_{\substack{\text{入射する光}\\\text{(放射輝度)}}} \underbrace{(\boldsymbol{\omega}'\cdot\mathbf{n})}_{\substack{\text{BRDFの}\\\text{コサイン}\\\text{成分}}} d\boldsymbol{\omega}'$$

BRDF(**双方向反射分布関数**)**とは**

ある一点(x)において入射(ω')した光が、どのように反射(ω)するかを表した関数

x : **出射が起こる座標**

$\boldsymbol{\omega}$: **出射方向**

$\boldsymbol{\omega}'$: **入射方向**

$\mathbf{n}$: x**の位置における法線**

レンダリング(3)

- 簡単にするためにBRDFをLambertのDiffuseと考えて…
  - 積分範囲Ωは全球方向とする
  - 入射する光(光源)は無限遠にある環境マップとする

$$\underbrace{L_o(x)}_{\text{出射される光 (Diffuseなので出射方向に依存しない)}} = \frac{1}{\underbrace{\pi}_{\text{Lambert DiffuseのBRDF}}} \int_\Omega \underbrace{L_i(\boldsymbol{\omega}')}_{\text{入射する光 (位置には依存しない)}} \underbrace{\max(\boldsymbol{\omega}'\cdot\mathbf{n}(x),0)}_{\text{BRDFのコサイン成分(全球の積分なので、正の範囲でクランプする)}} d\boldsymbol{\omega}'$$

Courtesy of Paul Debevec

レンダリング(4)

- 数値計算のために離散にすると…
  - sは半径1の球面上をサンプリングするベクトル
    - この場合は各ライトに対応する

$$L_o(x) = \frac{1}{\pi} \underbrace{\sum_{\mathbf{s}} L_i(\mathbf{s}) \max(\mathbf{s}\cdot\mathbf{n}(x),0) \underbrace{\Delta s}_{\text{sの方向の球面上の微小面積(立体角)}}}_{\text{ライトの数だけ処理する。sは各ライトのベクトル}}$$

レンダリング(5)

- ここでLambertのDiffuseとBRDFのコサイン部分を基底関数B(x)に投影すると…

$$\frac{1}{\pi}\max(\mathbf{s}\cdot\mathbf{n}(x),0)\Delta s \approx \underbrace{\sum_j c_j(x)B_j(\mathbf{s})}_{\text{cが投影された係数}}$$

$$L_o(x) = \sum_{\mathbf{s}}\left(L_i(\mathbf{s})\underbrace{\sum_j c_j(x)B_j(\mathbf{s})}\right)$$

代入する。係数cは位置ごとに存在する。例えば、頂点単位

レンダリング(6)

- 結果として
  - 係数cと基底関数Bの内積をライトの強度と掛けたものが明るさ(色)になる
    - ライトの数だけ繰り返す
  - 計算量は係数の数×ライトの数
    - sに対しての基底関数Bは計算で求めるか、テクスチャルックアップをする

$$L_o(x) = \sum_{\mathbf{s}} \left(L_i(\mathbf{s}) \sum_j c_j(x) B_j(\mathbf{s}) \right)$$

レンダリング(7)

- PRT計算のバリエーションとして
  - 遮蔽(影)を表現

$$\sum_j c_j(x)B_j(\mathbf{s}) \approx \int \frac{1}{\pi}\max(\mathbf{s}\cdot\mathbf{n}(x),0)\underbrace{V(\mathbf{s},x)}\,ds$$

位置xにおける方向sが遮蔽している場合は0、していない場合は1になるvisibility関数

  - 他にも前計算時に相互反射(interreflection)や表面下散乱(subsurface scattering)を計算することができる
    - 追加レンダリングコストなしで、これらの表現を実現できる

Image Based Lighting(1)

- ライトをImage Based Lighting(IBL)にしたい
  - IBLとは光源をテクスチャであらわすこと
    - テクセルの数だけ光源があると考えればよい
    - 例えば、128x128×6なら98,304個のライト!
  - まともに計算はできない

Image Based Lighting(2)

- IBL用の環境マップ(テクスチャ)も圧縮してみる
  - この場合のベクトルsはキューブマップの場合の(u,v,w)と考えればよい

$$\underbrace{\sum_k c_k B_k(\mathbf{s})} \approx \underbrace{L_i(\mathbf{s})}$$

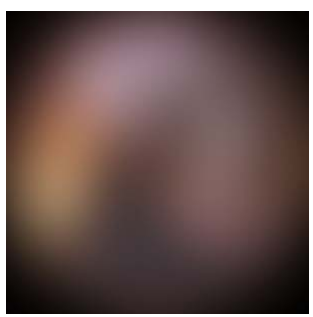
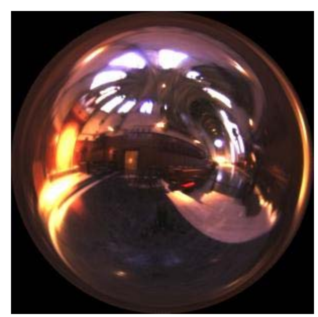



Image Based Lighting(3)

- 代入すると…

$$L_o(x) = \sum_{\mathbf{s}} \left(\underbrace{\sum_k c_k^{light} B_k(\mathbf{s})}_{\text{圧縮された光源}} \underbrace{\sum_j c_j^{prt}(x) B_j(\mathbf{s})}_{\text{圧縮されたPRTデータ}} \right)$$

Image Based Lighting(4)

- 計算を効率化するための変形を行なう

$$L_o(x) = \sum_{\mathbf{s}} \left(\sum_k c_k^{light} B_k(\mathbf{s}) \sum_j c_j^{prt}(x) B_j(\mathbf{s}) \right)$$

$$= \sum_k \sum_j \left(c_k^{light} c_j^{prt}(x) \underbrace{\sum_{\mathbf{s}} B_k(\mathbf{s}) B_j(\mathbf{s})} \right)$$

ここは基底関数を円周の全方向分足したもの(積分)なので、定数の行列にすることができる

Image Based Lighting(5)

- さらに変形する

$$\mathbf{A}_{kj} = \sum_{\mathbf{s}} B_k(\mathbf{s}) B_j(\mathbf{s})$$

$$L_o(x) = \sum_k \sum_j \mathbf{A}_{kj} c_k^{light} c_j^{prt}(x)$$

- 行列Aは事前に計算できる
  - 計算量は光源の係数の数×PRTデータの係数の数になる

さらなる効率化(1)

- 例えば係数が16個だとすると256回の積和が必要になってしまう
  - もっと効率化は出来ないか?
  - 式を眺めてみる

$$\mathbf{A}_{kj} = \sum_{\mathbf{s}} B_k(\mathbf{s}) B_j(\mathbf{s})$$

$$L_o(x) = \sum_k \sum_j \mathbf{A}_{kj} c_k^{light} c_j^{prt}(x)$$

さらなる効率化(2)

- 基底関数が直交関数の場合を考える

$$\mathbf{A}_{kj}=\sum_{\mathbf{s}}B_k(\mathbf{s})B_j(\mathbf{s})=0(k\neq j)$$

  - つまり、直交の場合は行列Aは対角成分のみしか残らない
    - 正規直交なら単位行列になる

さらなる効率化(3)

- 最終的に以下のようになる

$$L_o(x) = \sum_k \mathbf{A}_k c_k^{light} c_k^{prt}(x)$$

  - 正規直交の場合はAの項も必要ない
  - 計算量は係数の数だけになる
    - 係数16個なら16回の積和

シェーディング例

- たとえばSH関数を基底にした場合のDiffuseのレンダリング

```
Vector vOutputColor(0.0f, 0.0f, 0.0f);
for(int i = 0; i < N; i++) {
  vOutputColor.x += vPrt[nVertex][i].x * vLight[i].x;
  vOutputColor.y += vPrt[nVertex][i].y * vLight[i].y;
  vOutputColor.z += vPrt[nVertex][i].z * vLight[i].z;
}
```

スペキュラー

- スペキュラのシェーディングはどうするか?
  - 例えばSHでglossyの場合
    - データの計算法などは論文(Sloan 2002)をご覧ください

$$L_o(x) = \sum_i \underbrace{\alpha_i G_i^*}_{\text{畳み込みされたスペキュラー(BRDF)のフィルター係数}} \left(\sum_j \underbrace{(M(x))_{ij}}_{\text{圧縮されたPRTデータ(Transfer Matrix)}} \underbrace{L_j}_{\text{光源の係数}} \right) \underbrace{y_i(R)}_{\text{視線方向のSH基底}}$$

レンダリングまとめ

- 基底変換をした式をレンダリング方程式に当てはめると…
  - 内積計算でレンダリングできる
    - 現在のGPUに非常に向いている
    - ライト(IBL)も基底変換(圧縮)するとさらに計算はシンプルに
  - 直交関数系を利用すれば、大幅に計算量を減らすことができる

Outline

1. PRT とは?
2. 基底変換とデータ圧縮
3. レンダリング
4. PCA
    1. 概要
    2. PCAとは
    3. PCAの性質

実際の圧縮効率

- Spherical Harmonics
  - いわゆるフーリエ変換に相当する
    - 低周波数の係数のみをデータとして保持する
    - 係数の個数は事前に決める
      - たとえばマテリアルごとに
    - 高周波のデータを再現できない
      - たとえばエッジのするどい影やスペキュラー
  - Jpeg的な圧縮方法と言える

適応的な圧縮(非線形近似)

- SHがJpeg的な圧縮ならMP3のような圧縮はできないのか?
  - MP3のアルゴリズムを単純に言うと…
    - 人間に聞こえずらい周波数の音の量子化ビット数を減らしている
    - ビット数も可変だが、どの周波数の情報を落とすかも、音の状況にあわせて変えている

MP3的なPRT

- すでにいくつかの非線形近似手法は存在する
  - SH+CPCA
  - Wavelet
  - SRBF+CTA
- ここでは、CPCAなどにつながる基礎であるPCAについて触れる

PCA

- Principal Component Analysis(主成分分析)
  - データの重要度を解析する手法の一つ
    - 解析を行い重要なデータのみを残すことによりデータを圧縮する
    - SHの低周波数を一律残すわけでなく、重要な部分に重みをつけて残すような感覚

PCAの手法

- PCA手順
  1. n次元のベクトルN個からn×nの共分散行列Aを作る
  2. Aを固有値分解する
  3. 結果(固有値および固有ベクトル)を分析する
- 計算自体はライブラリが存在する
  - たとえばLAPACKなど
- ここでは計算方法ではなくPCAがどういう意味を持っているのかを考える

Outline

1. PRTとは?
2. 基底変換とデータ圧縮
3. レンダリング
4. PCA
   1. 概要
   2. PCAとは
   3. PCAの性質

分散

- データがどのくらいばらついているかを表す指標の一つ
  - データの各値からデータ全体の平均を引いた値の二乗和の平均
    - 値が大きいほど、そのデータは平均値からデータがばらついているということを表す

$$d = \frac{1}{N}\sum_{k}^{N}(g_k - \overline{g})^2$$

固有値(ベクトル)

- ある行列Aがあるとき
  - そのAでベクトルxを一次変換したときにxをλ倍したものと等しい場合
    - xをAの固有ベクトル(eigenvector)という
    - λをAの固有値(eigenvalue)と呼ぶ
  - これだけではどんな性質をもっているのか良くわからない

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

$$(\mathbf{x} \neq 0)$$

固有値分解(1)

- PCAなどを行なう行列Aは対称行列を扱う
  - 対称行列の固有値、固有ベクトルは実数になる
    - コンピュータで扱いやすい

固有値分解(2)

- 行列Aがn×nの対称行列のとき
  - そのn個固有値λと固有ベクトルuを使い行列DとUを作る

$$\mathbf{D}=\begin{pmatrix}\lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n\end{pmatrix}\quad \mathbf{U}=\begin{pmatrix}\mathbf{u}_1\\ \mathbf{u}_2\\ \vdots\\ \mathbf{u}_n\end{pmatrix}$$

固有値分解(3)

- 行列Aを行列DとUを使い以下のように表すことを固有値分解と呼ぶ
  - 固有ベクトルuによる行列Uは正規直交系になる
  - スペクトル分解など他の呼び名もある

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^{\mathrm{T}}$$

共分散行列(1)

- PCAとは
  - **共分散行列Aを固有値分解することを指す**
    - 共分散行列とは?

共分散行列(2)

- N個の3次元ベクトルrの共分散行列を考えてみる

$$\mathbf{r}_n = (x_n, y_n, z_n)$$

のとき、その平均を

$$\bar{\mathbf{r}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{r}_n$$

とする

共分散行列(3)

- 次にその平均を各ベクトルから引く
  - 原点 $\bar{\mathbf{r}}$ の座標系に平行移動する

$$\mathbf{c}_n = \mathbf{r}_n - \bar{\mathbf{r}}$$

- このベクトルcを利用して共分散行列Vを作る

$$\mathbf{V} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{c}_n\mathbf{c}_n^T$$

共分散行列(4)

- この共分散行列の中身を見ると

$$\mathbf{c}_n = \left(x_n^c, y_n^c, z_n^c\right)$$

のとき

$$\mathbf{V} = \frac{1}{N}\begin{pmatrix} \sum_{n=1}^{N}\left(x_n^c\right)^2 & \sum_{n=1}^{N} x_n^c y_n^c & \sum_{n=1}^{N} x_n^c z_n^c \\ \sum_{n=1}^{N} y_n^c x_n^c & \sum_{n=1}^{N}\left(y_n^c\right)^2 & \sum_{n=1}^{N} y_n^c z_n^c \\ \sum_{n=1}^{N} z_n^c x_n^c & \sum_{n=1}^{N} z_n^c y_n^c & \sum_{n=1}^{N}\left(z_n^c\right)^2 \end{pmatrix}$$

となる

Outline

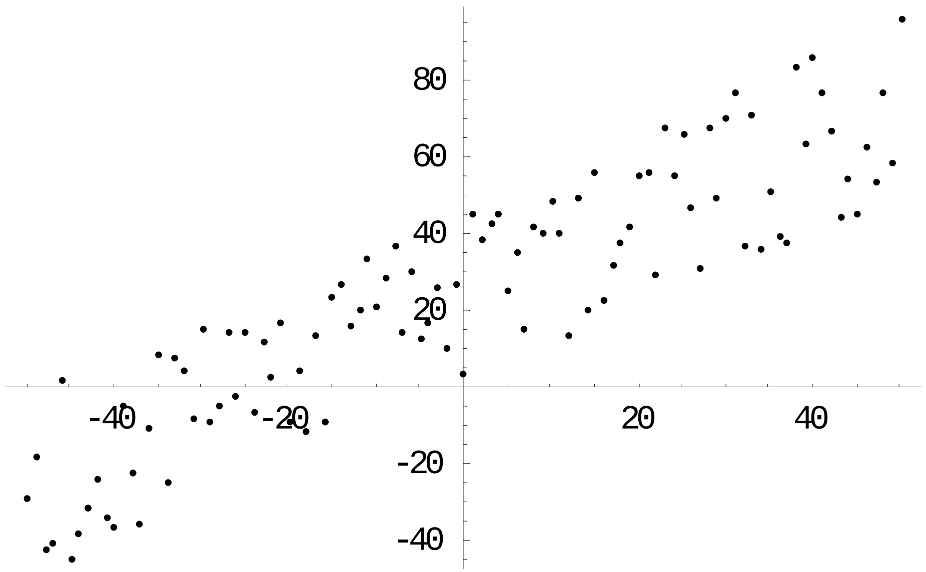
1. PRTとは?
2. 基底変換とデータ圧縮
3. レンダリング
4. PCA
    1. 概要
    2. PCAとは
    3. PCAの性質

PCAの性質(1)

- 共分散行列Aを固有値分解すると、どんな性質があるのか?
  - 固有値は行列Aの各固有ベクトル方向の分散を表している
  - 固有ベクトルで構成された行列Uは正規直交行列なので、回転行列になっている
    - この行列で回転すると、それぞれの軸に投影した値の分散が固有値になっていると言える

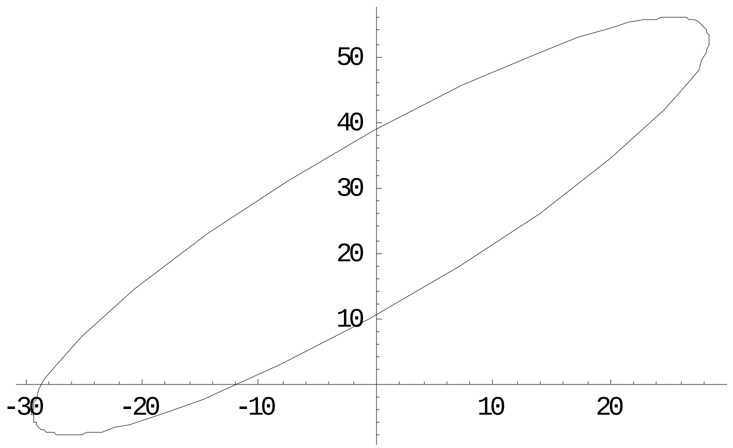
PCAの性質(2)

- 以下のような2次元のデータ列があった場合



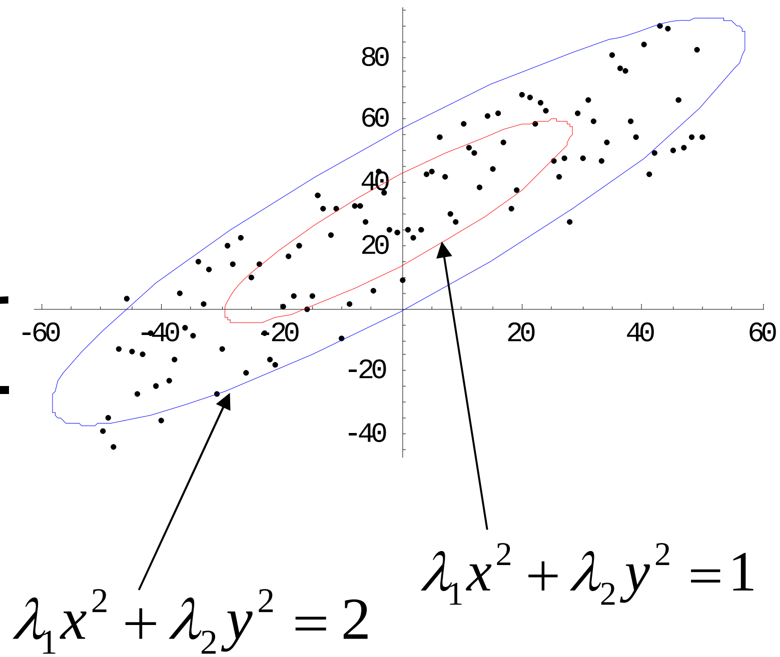
PCAの性質(3)

- このデータ列の共分散行列をAとしたときに、以下のようなグラフを描いてみる
  - この楕円は、$\lambda_1 x^2 + \lambda_2 y^2 = 1$ をAを固有値分解した場合の固有ベクトルの行列Uで回転した形になっている
    - λは各固有値



PCAの性質(4)

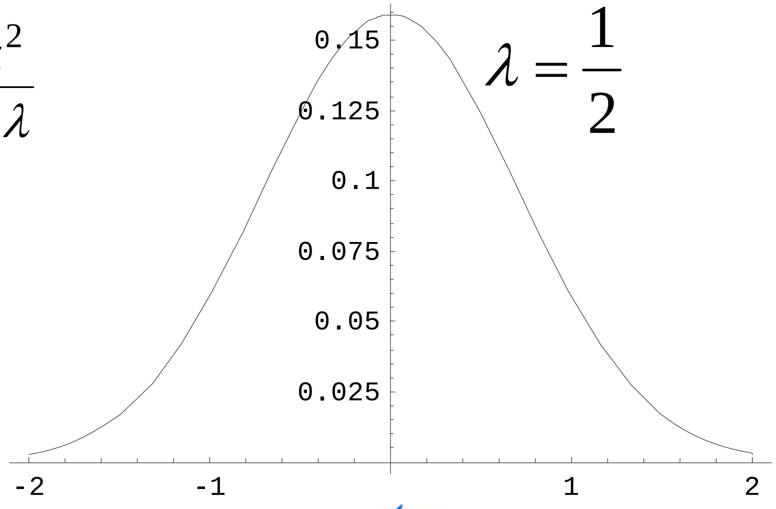
- このニつをグラフを重ねてみる
  - $\lambda_1 x^2 + \lambda_2 y^2 = 2$を回転したものも重ねてある
- 一番大きい固有値に対応する固有(PCA)ベクトルはデータのもっとも分散の大きい方向に対応し、固有値はその分散を表している



PCAの性質(5)

- 各ベクトルを対応するPCAベクトルに投影すると..
  - 固有値は正規分布(ガウス分布)のパラメータになっていて、データの出現頻度を表している

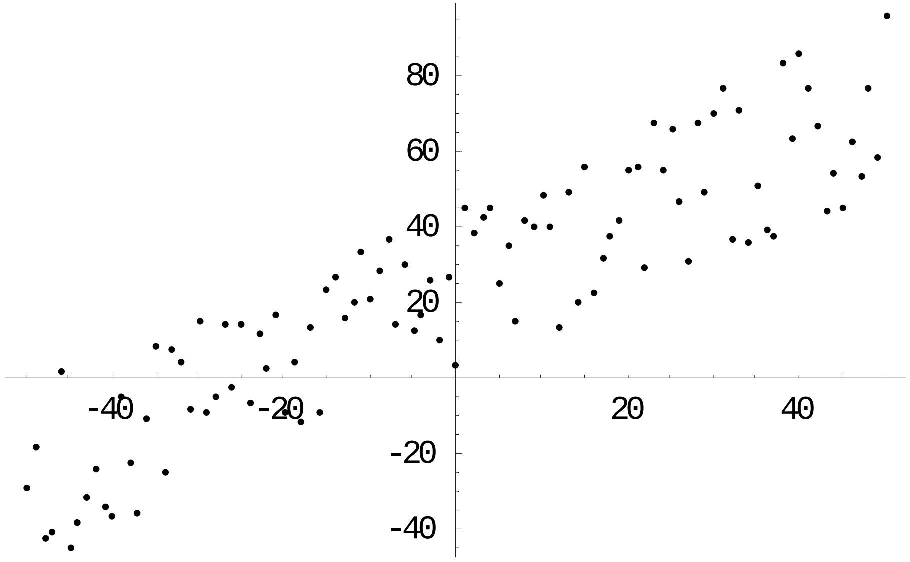
$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2\lambda}}$$



$$\lambda = \frac{1}{2}$$

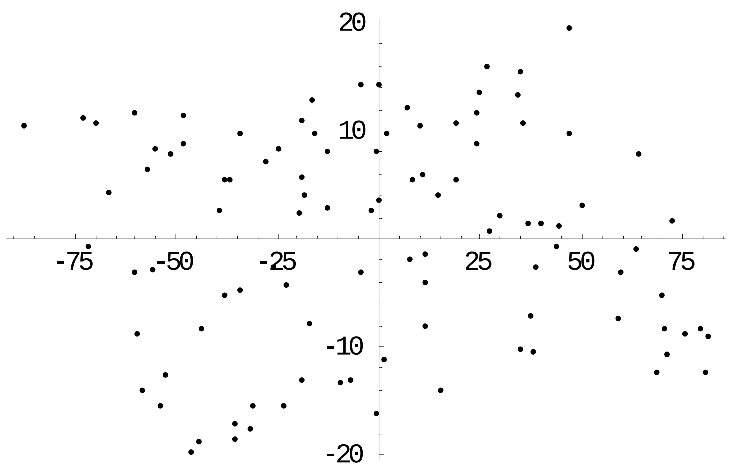
PCAの効果(1)

- 実際にPCAの意味を確かめてみる
  - 以下のデータは(-50,-44.14)-(50,96.38)に分布している



PCAの効果(2)

- 回転してみると…
  - (-87.52, -19.61)-(81.21,19.54)の範囲に収まっている
    - 面積で考えると、元データは14052で回転後は6605.8になっている
    - 整数部の量子化bit数で考えると、x軸では7bitから8bitへ、y軸は8bitから6bitへ(トータルで1bit減)



PCAの効果(3)

- PCAを行なうとそのベクトル列の最も分散の大きい方向がわかる
  - 値のバラつきが多い方向ということなので、その軸方向にデータがよく使われているということになる
    - この軸を主軸と呼ぶ
  - 例) そのベクトル列を囲む、タイトな四角形を求めることもできる(最適なBounding Boxではない)
  - 例) 多次元データの検索をシンプルに高速化するために、ある方向(一軸)にソートする場合、最もソート効率の高くなる方向がわかる

PCA圧縮

- n次元ベクトル列xが存在するときに、そのxをPCAすると
  - データに偏りがあると最初の数個の固有値が大きく、あとの残りはほとんど0になる
    - 固有値(分散)がほぼ0ということは、その軸上でのデータの値は、ほぼ定数
  - つまり、この次元(軸)はそのベクトルには必要ないことがわかる
    - 必要な次元のデータのみ出力する

PCA圧縮まとめ

- PCA圧縮とは
  - ベクトル(データ)列に対してPCAを行なう
  - 固有ベクトルで作られた行列で回転する
  - 固有値の大きい順(または閾値)で必要な次元 (その固有値に対応した軸)のデータのみ保存する

PCAの高速化

- PCAの計算は重い
  - 大きい次元のデータのPCAは負荷が高い
    - ほんとんどの場合は、一部の固有値の大きいデータを求めれば充分
  - 特異値分解(Singular Value Decomposition, SVD)を利用できる
    - ライブラリが存在するので、計算方法を気にする必要はない

まとめ(1)

- PRTを利用すると限定的なGlobal Illuminationをリアルタイムで実現できる
  - InterreflectionやSubsurface Scatteringなどの次世代的なエフェクト
    - Diffuseだけならコスト的にも見合う
  - よりダイナミックなアニメーションは現在も研究中
    - いくつかの論文は現実的になりつつある
- しかしPRTデータ量が大きいので…

まとめ(2)

- 基底変換を利用してデータを圧縮
  - 基底変換には最小二乗法を利用する
    - 基底が直交関数系であれば、直交変換を利用できるので計算量を大幅に減らすことができる
    - レンダリングも高速化できる
      - 光源も圧縮すればさらに速くなる
    - DC部分を利用すればAmbient Occlusionになる
    - より効率的な基底などはまだまだ研究中
- それでも、まだデータが大きい場合がある
  - シャープなシャドウを実現したい場合
  - Specular(glossy)を再現したい場合

まとめ(3)

- PCAを利用する
  - 実際にはクラスタに分ける
    - 格段に圧縮効率があがる
    - データ量が減って、クオリティもあがる
  - CTAなどもある
    - PCAよりも計算量が増えるが、よりドラスティックな圧縮が可能
    - このような圧縮もまだまだ研究中

参考文献

- Peter-Pike Sloan et al. “Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments.” SIGGRAPH 2002
- Peter-Pike Sloan et al. “Clustered Principal Components for Precomputed Radiance Transfer” SIGGRAPH 2003
- Peter-Pike Sloan et al. “All-Frequency Precomputed Radiance Transfer for Glossy Objects” EGSR 2004
- **金谷健一著「これなら分かる応用数学教室」共立出版**

謝辞

- トライエース研究開発部スタッフ
- (株)マイクロソフト Peter-Pike Sloan氏
- (株)ピラミッド 田村尚希氏

ご質問は?

- このスライドは以下のサイトでダウンロードすることができます
  - http://research.tri-ace.com
- ご質問などは以下のアドレスまで
  - research@tri-ace.co.jp

Appendix – このスライドでのレンダリング方程式
記号その1

記号	意味
L_o	反射された放射輝度(光の強さ)
L_e	自ら発光している放射輝度
f_r	BRDF関数
L_i	入射した光の放射輝度
ω	反射された光の方向を表すベクトル
ω'	入射した光の方向を表すベクトル
n	xにおける法線ベクトル
x	物体の表面上のある一点を表す変数

Appendix – このスライドでのレンダリング方程式
記号その2

記号	意味
s	ライトや数値積分時のサンプリング方向を表すベクトル
V	遮蔽関数。値が0の時には遮蔽されていて、1の時に遮蔽されていないことを表す
Δs	離散積分の時のs方向の球面上の微小面積(立体角)
c_j	基底変換した係数ベクトル(light,prtバージョンもあり)
B_j	基底関数