白黒→カラー化
Colorization

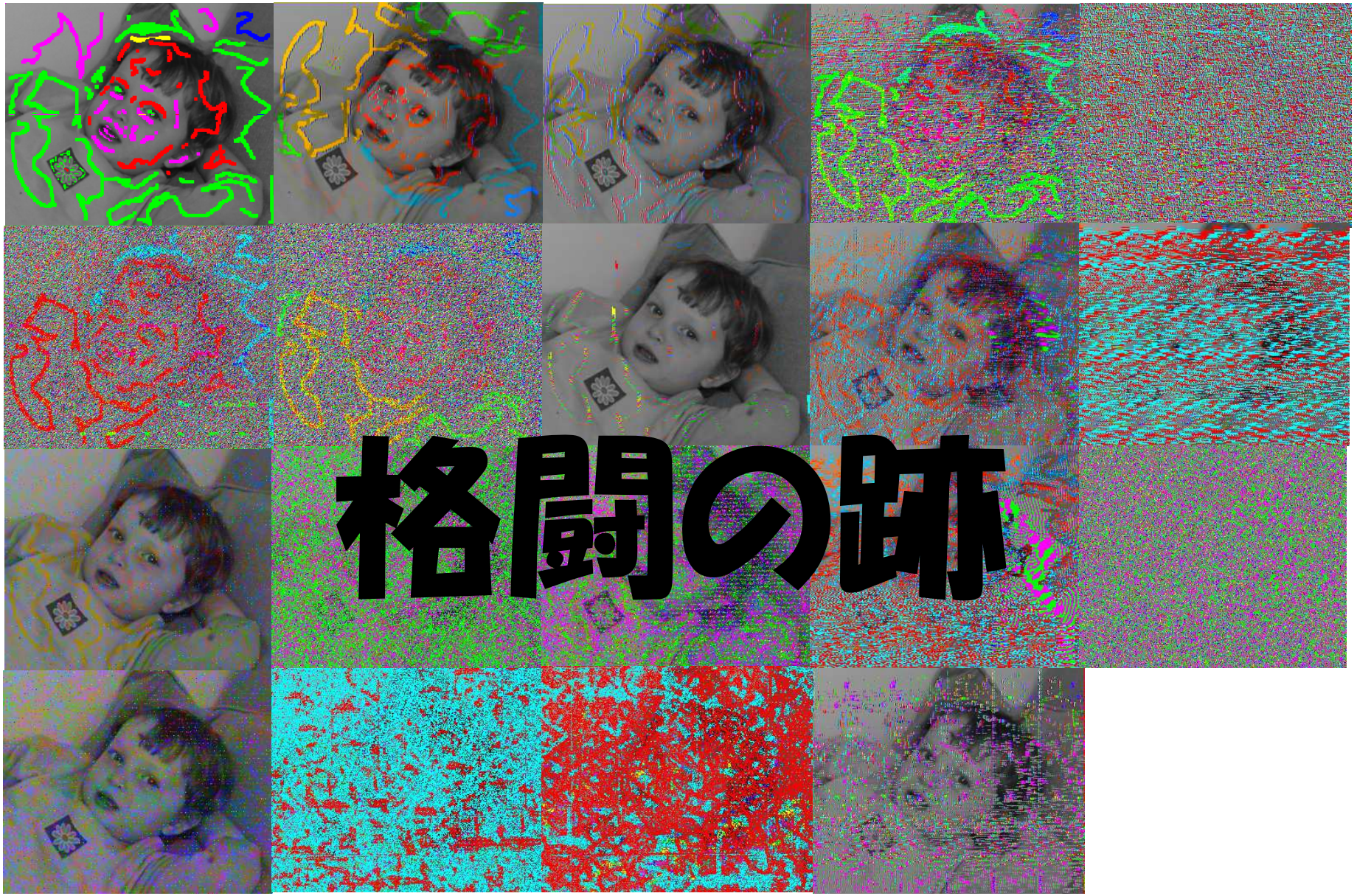
Colorization Using Optimization

Siggraph 2004論文の実装

Colorization using Optimization

Anat Levin Dani Lischinski Yair Weiss

School of Computer Science and Engineering
The Hebrew University of Jerusalem*


格闘の跡

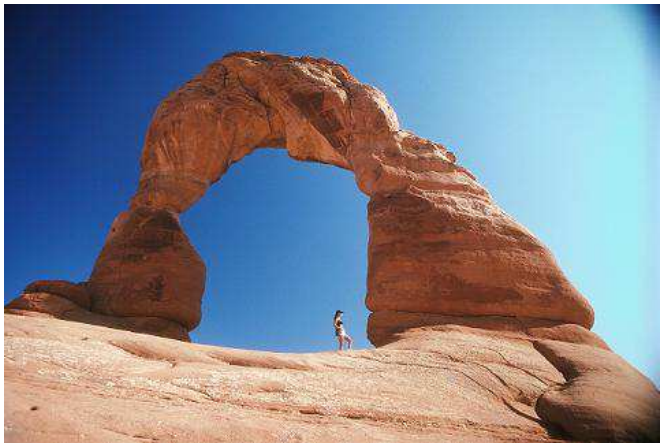
Conjugate Gradient Method for a Sparse Systemでやっと
収束計算が成功！！

Numerical Recipes
「**http://www.aip.de/groups/soe/local/numres/**」
「**http://www.aip.de/groups/soe/local/numres/bookcpdf/**」

2.7 Sparse Linear Systemsを参照

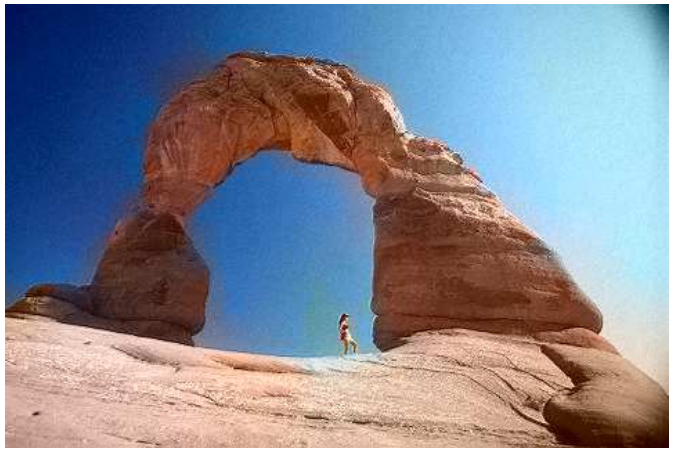
論文Levinらの結果

論文Levinらの結果



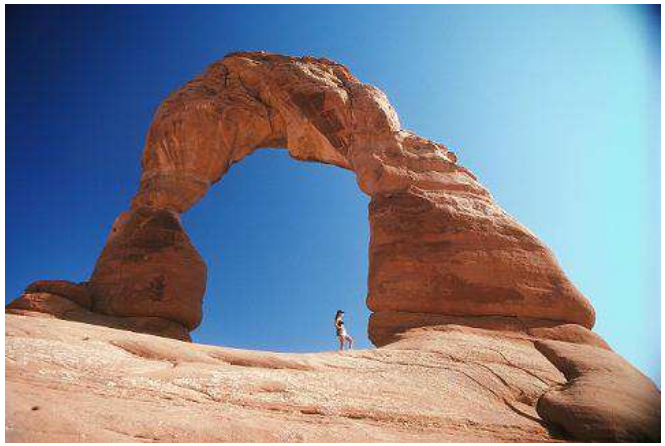
ピクセル値(0～1)への正規化を止めた

320x265 計算時間　3分->スレッド化 1分



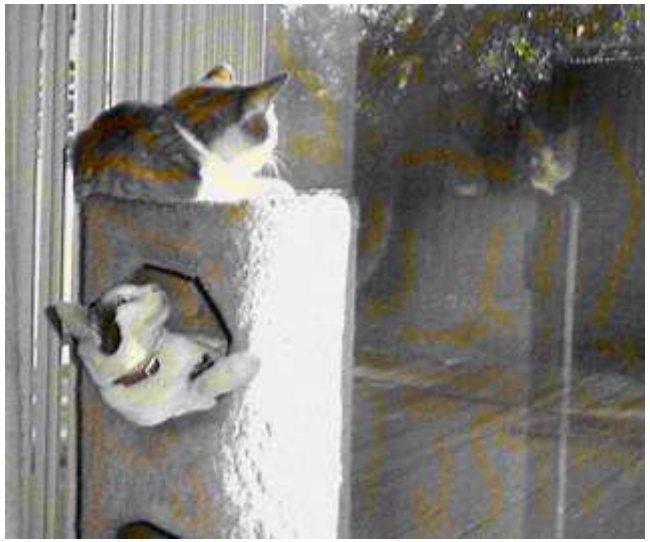
論文Levinらの結果

論文Levinらの結果



http://www.cs.huji.ac.il/~yweiss/Colorization/

結果



論文Levinらの結果

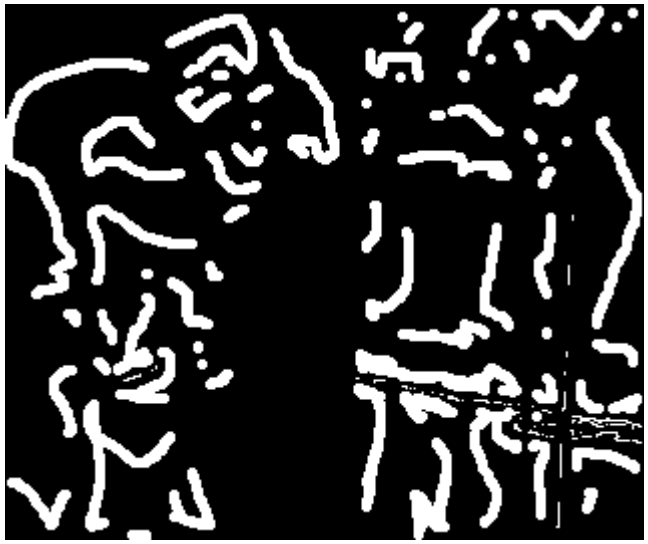
白黒画像 - マスク画像

これは「おかしい」
原因は背景の白黒部分にも
差があるため拘束条件となっ
てしまっている。

実際、拘束されている場所（白）
は以下のようになっている。

白黒画像 - マスク画像

R=G=Bの部分は白黒なので元の白黒画像に合わせてやる。

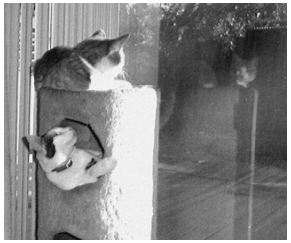


拘束条件が正しくなった。

結果

論文Levinらの結果

白黒画像



カラー化結果

論文Levinらの結果

白黒画像

カラー化結果

論文Levinらの結果

ものすごく時間のかかる処理

```
int dsprsin(double **a, int n, double thresh, unsigned long nmax, double sa[], unsigned long ija[])
{
	int i,j;
	unsigned long k;
	for (j=1;j<=n;j++) sa[j] = a[j][j];
	ija[1]=n+2;
	k=n+1;
	for (i=1;i<=n;i++) {//Loop over rows.
		for (j=1;j<=n;j++) {//Loop over columns.
			if (fabs(a[i][j]) >= thresh && i != j) {
				if (++k > nmax) return -1;
				sa[k]=a[i][j];
				ija[k]=j;
			}
		}
		ija[i+1]=k+1;//As each row is completed, store index to
	}
	return 0;
}
```

このコードは仕組み説明にしかならない。
a[i][j]のような2次元配列ではメモリ確保出来ないから。
200x200の画像の場合本処理のマトリクスはRGB=3x4バイトとして
(200x200)x(200x200)x3x4=**18Gバイト**

Numerical Recipes
「**http://www.aip.de/groups/soe/local/numres/bookcpdf/**」

2.7 Sparse Linear Systemsを参照

行列の圧縮

$$\begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 3 & 0 & 4 \\ 5 & 0 & 6 & 7 \\ 0 & 8 & 0 & 9 \end{bmatrix}$$

$$\begin{aligned} \mathrm{val} &= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix} \\ \mathrm{col} &= \begin{bmatrix} 0 & 2 & 1 & 3 & 0 & 2 & 3 & 1 & 3 \end{bmatrix} \\ \mathrm{row} &= \begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 3 \end{bmatrix} \end{aligned}$$

非ゼロ要素だけ覚えておく。

A[row[0]][col[0]]=1
A[row[1]][col[1]]=2
A[row[8]][col[8]]=9
のようにアクセスできる。

Val, col, row配列を作ったとしてさっきのコードを書き直す。

最初の部分だけをみてみよう。

for (j=1;j<=n;j++) sa[j] = **a[j][j];**

これは対角成分だけを設定している。
しかし、ここで、、、、、

$$\begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 3 & 0 & 4 \\ 5 & 0 & 6 & 7 \\ 0 & 8 & 0 & 9 \end{bmatrix}$$

$$\mathrm{val} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$
$$\mathrm{col} = \begin{bmatrix} 0 & 2 & 1 & 3 & 0 & 2 & 3 & 1 & 3 \end{bmatrix}$$
$$\mathrm{row} = \begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 3 \end{bmatrix}$$

せっかく行列をval,col,rowで圧縮管理しできたが
この単純なコードを書き直すことが出来ない。
出来るが以下のようになる。

```
for (j=1;j<=n;j++)
{
   for ( int kk = 0; kk < N; kk++ )
   {
        if ( col[kk] != j && row[kk] != j ) continue;
        sa[j] = val[kk];
    }
}
```

行と列がjの要素を探さないと、、、、

間違ってはいないがこの方法だと500x500画像だと絶望的な処理時間になる。
※処理が戻ってこない、、、永遠に思えるほど。

Indexed Storageをするにはi,j(行と列）を指定した時に圧縮形式から素早く対応要素を特定する必要がある。sa, ija配列に設定してしまえば後は何も要らなくなる。

とりあえず、今の実装はstd:pairとstd:mapを使ってある程度は高速化できたがまだ遅い。

```
class SparseMatrix
{
	std::map<std::pair<int, int>, int> sparseMatrix;

public:
	SparseMatrix(){}
	void set( int val,  int row,  int col)
	{
	   sparseMatrix[std::pair<int, int>(row, col)] = val;
	}
	int get( int row,  int col) const
	{
		const std::pair<int, int> key = std::pair<int, int>(row, col);

		 std::map<std::pair<int, int>, int>::const_iterator ret =  sparseMatrix.find(key);

		 if (ret != sparseMatrix.end())
		 {
		     return ret->second;
		 }
		 return -1;
	 }
};
```

こうやってpairとmapで管理しておく。

```
SparseMatrix sp;
for (int kk = 0; k k< N; kk++)
{
   sp.set(kk, row[kk], col[kk]);
}
```

```
for (j=1;j<=n;j++) sa[j] = a[j][j];
```

```
for (j=1;j<=n;j++)
{
   for ( int kk = 0; kk < N; kk++ )
   {
        if ( col[kk] != j && row[kk] != j ) continue;
        sa[j] = val[kk];
    }
}
```

```
for (j=1;j<=n;j++)
{
   int kk = sp.get( j, j);
   if ( kk >= 0) sa[j] = val[kk];
}
```