Wavelet Radiosity Operations

Andrew Willmott (*ajw+@cs.cmu.edu*)

Introduction

In this note we give a brief introduction to how the push, pull, and inter-element radiosity transfer operations can be carried out in wavelet radiosity. We then list the coefficients required for these operations for a number of different bases, over both quadrilaterals and triangles. Finally we give some examples of how to use these coefficients.

Standard Bases

To 'push' radiosity from a patch to one of its children requires a single linear transformation, and thus can be accomplished by a matrix multiply. Generally, if $\mathbf{b}$ is a vector holding the basis coefficients of the radiosity function for the parent patch, and $\mathbf{b}_i$ holds the coefficients for the ith child patch, then

$$\mathbf{b}_i = P_i \mathbf{b}. \tag{1}$$

where P_i are square matrices. The 'pull' operation can be written as a sum of similar operations on the child basis coefficients:

$$\mathbf{b} = \sum_i L_i \mathbf{b}_i. \tag{2}$$

We call P_i the push matrices, and L_i the pull matrices; they are different for different bases, and different parametrizations of those bases.

To calculate the transfer of radiosity between two patches, the source patch having coefficients $\mathbf{b}^s$, and the destination $\mathbf{b}^d$, we need a transfer matrix T. This can be calculated from a matrix of kernel samples from the source to the destination patches, K, and two other matrices, as follows:

$$T = RKS \tag{3}$$

Here S transforms the source coefficients to quadrature samples on the source patch, K transforms these to the equivalent samples on the receiving patch, and R projects these into the destination basis. We then have:

$$\mathbf{b}^d = T\mathbf{b}^s. \tag{4}$$

We calculate K as follows:

$$K_{ij} = \kappa(\text{map}^s(\mathbf{u}_i), \text{map}^d(\mathbf{u}_j)), \tag{5}$$

where κ is defined as

$$\kappa(x, x') = \frac{\cos\theta \cos\theta'}{\pi r^2}, \tag{6}$$

and the 'map' functions map from the parameter space of **u** to a point **x** on the source or destination patches. The parametric points **u** correspond to the quadrature points used to calculate *R* and *S*.

Tensor-Product Bases

For wavelet radiosity over rectangular patches we can use the tensor product of 1D bases to cover a 2D surface, so that $f(x,y) = g(x)g(y)$. We store our basis coefficients in an $n \times n$ matrix, *B*, where *n* is the order of the basis, and our push is accomplished by a tensor matrix multiplication as follows:

$$B_i = P_i B P_i^T . \tag{7}$$

Similarly, for a 'pull' operation from all the children of a patch to the parent patch, we have:

$$B = \sum_i L_i B_i L_i^T \tag{8}$$

For tensor-product bases, *K* and *T* become four-dimensional matrices, and we calculate *K* as follows:

$$K_{ijkl} = \kappa(\text{map}(u_i, u_j), \text{map}(u_k, u_l)) \quad . \tag{9}$$

Calculating *T* from *K* is not quite as straightforward as a pair of tensor matrix multiplications, because, while *S* and *R* both represent 1D linear transformations, *K* represents a full, 2D linear transformation[1]. Instead, we find that:

$$T_{ijkl} = \sum_s \sum_t S_{is} S_{jt} \left(\sum_u \sum_v R_{ku} R_{lv} K_{stuv} \right). \tag{10}$$

To take advantage of existing 2D matrix-multiply routines, this can be calculated in two stages:

$$T'_{ij} = R K_{ij} R^T , \tag{11}$$

and then,

$$T_{ij} = \sum_s \sum_t S_{is} S_{jt} T'_{st} . \tag{12}$$

1. In fact, we could store *R* and *S* as full, 2D linear transformations, so that calculating the transfer coefficients *could* be achieved by a couple of simple matrix multiplications; these would be 4×4 matrix multiplies for the M2 basis, for example. While this would be simpler conceptually, and for coding, it would take more arithmetic operations than using the (more involved) tensor multiplications, so we recommend the above approach.

Note that T_{ij}, T'_{st}, and R are $n \times n$ matrices, and S_{is} and S_{jt} are scalars.

Finally, from T the transfer of radiosity can be calculated as follows:

$$B^d = \sum T_{ij} B^s_{ij} \tag{13}$$

The Coefficients

Coefficients used for wavelet radiosity are listed below. The first part of the name denotes the basis, and the second part the purpose: P for push matrix, L for pull matrix, R for kernel-to-destination matrix, S for source-to-kernel matrix, and U for the quadrature sample position matrix. For the flatlet bases, R and S are both the identity matrix, and thus $T = K$. The bases these coefficients describe are all parameterized over the range [0,1].

(These coefficients are also available in text file format from *http://www.cs.cmu.edu/~radiosity/dist/WaveCoeffs.h*)

```
// === Quad (Tensor) Push/Pull Matrices ======================================

M2_LINE_P0 = [1.0, 0.0]
        [0.0, 0.5]
M2_LINE_P1 = [1.0, 0.5]
        [0.0, 0.5]
M2_LINE_L0 = [1.25, 0.5]
        [-1.5, -0.5]
M2_LINE_L1 = [-0.25, -0.25]
        [1.5, 1.0]

F2_LINE_P0 = [1.25, -0.25]
        [0.75, 0.25]
F2_LINE_P1 = [0.25, 0.75]
        [-0.25, 1.25]
F2_LINE_L0 = [0.5625, 0.3125]
        [-0.0625, 0.1875]
F2_LINE_L1 = [0.1875, -0.0625]
        [0.3125, 0.5625]

M3_LINE_P0 = [1.0, 0.0, -0.25]
        [0.0, 0.5, 0.0]
        [0.0, 0.0, 0.25]
M3_LINE_P1 = [1.0, 0.5, 0.0]
        [0.0, 0.5, 0.5]
        [0.0, 0.0, 0.25]
M3_LINE_L0 = [1.25, -0.4375, -1]
        [-1.5, 1.375, 2.0]
        [0.0, -1.875, -1.75]
M3_LINE_L1 = [-0.25, 0.6875, 0.875]
        [1.5, -0.875, -1.75]
        [0.0, 1.875, 2.0]

F3_LINE_P0 = [1.375, -0.5, 0.125]
        [0.625, 0.5, -0.125]
        [0.125, 1.0, -0.125]
```

```
F3_LINE_P1 = [-0.125, 1.0, 0.125]
        [-0.125, 0.5, 0.625]
        [0.125, -0.5, 1.375]
F3_LINE_L0 = [0.583333, 0.291667, 0.125]
        [-0.0833333, 0.25, 0.333333]
        [0.0, -0.0416667, 0.0416667]
F3_LINE_L1 = [0.0416666, -0.0416666, 0.0]
        [0.333333, 0.25, -0.0833333]
        [0.125, 0.291667, 0.583333]

// === Quad (Tensor) Transfer Matrices ========================================

M2_LINE_R = [1.36603, -0.366025]
        [-1.73205, 1.73205]
M2_LINE_S = [0.5, 0.105662]
        [0.5, 0.394338]

M3_LINE_R = [2.58994, -2.88889, 1.29895]
        [-4.62433, 6.66667, -2.04234]
        [3.33333, -6.66667, 3.33333]
M3_LINE_S = [0.277778, 0.031306, -0.0890644]
        [0.444444, 0.222222, -0.037037]
        [0.277778, 0.246472, 0.126101]

M2_LINE_U = [0.211325, 0.211325]
        [0.788675, 0.211325]
        [0.211325, 0.788675]
        [0.788675, 0.788675]
M3_LINE_U = [0.112702, 0.112702]
        [0.5, 0.112702]
        [0.887298, 0.112702]
        [0.112702, 0.5]
        [0.5, 0.5]
        [0.887298, 0.5]
        [0.112702, 0.887298]
        [0.5, 0.887298]
        [0.887298, 0.887298]

// === Triangular Push/Pull Matrices ==========================================

M2_TRI_P0 = [1.0, 0.0, 0.0]
        [0.5, 0.5, 0.0]
        [0.5, 0.0, 0.5]
M2_TRI_P1 = [0.5, 0.5, 0.0]
        [0.0, 1.0, 0.0]
        [0.0, 0.5, 0.5]
M2_TRI_P2 = [0.5, 0.0, 0.5]
        [0.0, 0.5, 0.5]
        [0.0, 0.0, 1.0]
M2_TRI_P3 = [0.0, 0.5, 0.5]
        [0.5, 0.0, 0.5]
        [0.5, 0.5, 0.0]
M2_TRI_L0 = [0.5, 0.375, 0.375]
        [-0.125, 0.0, -0.125]
        [-0.125, -0.125, 0.0]
M2_TRI_L1 = [0.0, -0.125, -0.125]
        [0.375, 0.5, 0.375]
        [-0.125, -0.125, 0.0]
M2_TRI_L2 = [0.0, -0.125, -0.125]
```

```
        [-0.125, 0.0, -0.125]
        [0.375, 0.375, 0.5]
M2_TRI_L3 = [0.0, 0.125, 0.125]
        [0.125, 0.0, 0.125]
        [0.125, 0.125, 0.0]

F2_TRI_P0 = [1.6875, 0.1875, 0.1875, -1.0625]
        [0.6875, 0.1875, -0.3125, 0.4375]
        [0.6875, -0.3125, 0.1875, 0.4375]
        [0.9375, -0.0625, -0.0625, 0.1875]
F2_TRI_P1 = [0.1875, 0.6875, -0.3125, 0.4375]
        [0.1875, 1.6875, 0.1875, -1.0625]
        [-0.3125, 0.6875, 0.1875, 0.4375]
        [-0.0625, 0.9375, -0.0625, 0.1875]
F2_TRI_P2 = [0.1875, -0.3125, 0.6875, 0.4375]
        [-0.3125, 0.1875, 0.6875, 0.4375]
        [0.1875, 0.1875, 1.6875, -1.0625]
        [-0.0625, -0.0625, 0.9375, 0.1875]
F2_TRI_P3 = [-0.3125, 0.1875, 0.1875, 0.9375]
        [0.1875, -0.3125, 0.1875, 0.9375]
        [0.1875, 0.1875, -0.3125, 0.9375]
        [-0.0625, -0.0625, -0.0625, 1.1875]
F2_TRI_L0 = [0.317274, 0.140191, 0.140191, 0.178385]
        [0.00477431, 0.077691, -0.047309, -0.00911458]
        [0.00477431, -0.047309, 0.077691, -0.00911458]
        [-0.0768229, 0.0794271, 0.0794271, 0.0898438]
F2_TRI_L1 = [0.077691, 0.00477431, -0.047309, -0.00911458]
        [0.140191, 0.317274, 0.140191, 0.178385]
        [-0.047309, 0.00477431, 0.077691, -0.00911458]
        [0.0794271, -0.0768229, 0.0794271, 0.0898438]
F2_TRI_L2 = [0.077691, -0.047309, 0.00477431, -0.00911458]
        [-0.047309, 0.077691, 0.00477431, -0.00911458]
        [0.140191, 0.140191, 0.317274, 0.178385]
        [0.0794271, 0.0794271, -0.0768229, 0.0898438]
F2_TRI_L3 = [-0.0351562, 0.0898437, 0.0898438, 0.0273437]
        [0.0898438, -0.0351562, 0.0898438, 0.0273437]
        [0.0898438, 0.0898437, -0.0351563, 0.0273438]
        [0.105469, 0.105469, 0.105469, 0.167969]

M3_TRI_P0 = [1.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        [0.25, 0.25, 0.0, 0.5, 0.0, 0.0]
        [0.25, 0.0, 0.25, 0.0, 0.0, 0.5]
        [0.5, 0.0, 0.0, 0.5, 0.0, 0.0]
        [0.25, 0.0, 0.0, 0.25, 0.25, 0.25]
        [0.5, 0.0, 0.0, 0.0, 0.0, 0.5]
M3_TRI_P1 = [0.25, 0.25, 0.0, 0.5, 0.0, 0.0]
        [0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
        [0.0, 0.25, 0.25, 0.0, 0.5, 0.0]
        [0.0, 0.5, 0.0, 0.5, 0.0, 0.0]
        [0.0, 0.5, 0.0, 0.0, 0.5, 0.0]
        [0.0, 0.25, 0.0, 0.25, 0.25, 0.25]
M3_TRI_P2 = [0.25, 0.0, 0.25, 0.0, 0.0, 0.5]
        [0.0, 0.25, 0.25, 0.0, 0.5, 0.0]
        [0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
        [0.0, 0.0, 0.25, 0.25, 0.25, 0.25]
        [0.0, 0.0, 0.5, 0.0, 0.5, 0.0]
        [0.0, 0.0, 0.5, 0.0, 0.0, 0.5]
M3_TRI_P3 = [0.0, 0.25, 0.25, 0.0, 0.5, 0.0]
        [0.25, 0.0, 0.25, 0.0, 0.0, 0.5]
```

```
        [0.25, 0.25, 0.0, 0.5, 0.0, 0.0]
        [0.0, 0.0, 0.25, 0.25, 0.25, 0.25]
        [0.25, 0.0, 0.0, 0.25, 0.25, 0.25]
        [0.0, 0.25, 0.0, 0.25, 0.25, 0.25]
M3_TRI_L0 = [0.6875, 0.125, 0.125, 0.375, 0.125, 0.375]
        [0.0625, -0.125, 0.0625, -0.0625, -0.0625, 0.0625]
        [0.0625, 0.0625, -0.125, 0.0625, -0.0625, -0.0625]
        [-0.3125, 0.4375, -0.1875, 0.125, 0.125, -0.25]
        [0.0625, -0.0625, -0.0625, 0.0, 0.0, 0.0]
        [-0.3125, -0.1875, 0.4375, -0.25, 0.125, 0.125]
M3_TRI_L1 = [-0.125, 0.0625, 0.0625, -0.0625, 0.0625, -0.0625]
        [0.125, 0.6875, 0.125, 0.375, 0.375, 0.125]
        [0.0625, 0.0625, -0.125, 0.0625, -0.0625, -0.0625]
        [0.4375, -0.3125, -0.1875, 0.125, -0.25, 0.125]
        [-0.1875, -0.3125, 0.4375, -0.25, 0.125, 0.125]
        [-0.0625, 0.0625, -0.0625, 0.0, 0.0, 0.0]
M3_TRI_L2 = [-0.125, 0.0625, 0.0625, -0.0625, 0.0625, -0.0625]
        [0.0625, -0.125, 0.0625, -0.0625, -0.0625, 0.0625]
        [0.125, 0.125, 0.6875, 0.125, 0.375, 0.375]
        [-0.0625, -0.0625, 0.0625, 0.0, 0.0, 0.0]
        [-0.1875, 0.4375, -0.3125, 0.125, 0.125, -0.25]
        [0.4375, -0.1875, -0.3125, 0.125, -0.25, 0.125]
M3_TRI_L3 = [-0.0625, -0.125, -0.125, -0.125, -0.125, -0.125]
        [-0.125, -0.0625, -0.125, -0.125, -0.125, -0.125]
        [-0.125, -0.125, -0.0625, -0.125, -0.125, -0.125]
        [0.0625, 0.0625, 0.4375, 0.125, 0.25, 0.25]
        [0.4375, 0.0625, 0.0625, 0.25, 0.125, 0.25]
        [0.0625, 0.4375, 0.0625, 0.25, 0.25, 0.125]

F3_TRI_P0 = [1.91667, -0.427083, -0.114583, -0.427083, -0.583333, 0.822917, -0.114583,
0.822917, -0.895833]
        [0.59375, 0.90625, 0.125, -0.375, 0.28125, -0.9375, -0.03125, 0.15625, 0.28125]
        [0.104167, 1.57292, 0.0729167, 0.0104167, 0.354167, -0.864583, 0.0104167, -
0.177083, -0.0833333]
        [0.59375, -0.375, -0.03125, 0.90625, 0.28125, 0.15625, 0.125, -0.9375, 0.28125]
        [-0.0625, -0.125, 0.0, -0.125, -0.0625, 0.0625, 0.0, 0.0625, 1.25]
        [0.114583, 0.645833, 0.0520833, -0.260417, 0.177083, -0.510417, 0.0208333, -
0.0416667, 0.802083]
        [0.104167, 0.0104167, 0.0104167, 1.57292, 0.354167, -0.177083, 0.0729167, -
0.864583, -0.0833333]
        [0.114583, -0.260417, 0.0208333, 0.645833, 0.177083, -0.0416667, 0.0520833, -
0.510417, 0.802083]
        [0.895833, -0.104167, 0.0208333, -0.104167, 0.0208333, -0.0416667, 0.0208333, -
0.0416667, 0.333333]
F3_TRI_P1 = [0.0729167, 1.57292, 0.104167, 0.354167, 0.0104167, -0.0833333, 0.0104167,
-0.177083, -0.864583]
        [0.125, 0.90625, 0.59375, 0.28125, -0.375, 0.28125, -0.03125, 0.15625, -0.9375]
        [-0.114583, -0.427083, 1.91667, -0.583333, -0.427083, -0.895833, -0.114583,
0.822917, 0.822917]
        [0.0, -0.125, -0.0625, -0.0625, -0.125, 1.25, 0.0, 0.0625, 0.0625]
        [-0.03125, -0.375, 0.59375, 0.28125, 0.90625, 0.28125, 0.125, -0.9375, 0.15625]
        [0.0208333, -0.104167, 0.895833, 0.0208333, -0.104167, 0.333333, 0.0208333, -
0.0416667, -0.0416667]
        [0.0104167, 0.0104167, 0.104167, 0.354167, 1.57292, -0.0833333, 0.0729167, -
0.864583, -0.177083]
        [0.0208333, -0.260417, 0.114583, 0.177083, 0.645833, 0.802083, 0.0520833, -
0.510417, -0.0416667]
        [0.0520833, 0.645833, 0.114583, 0.177083, -0.260417, 0.802083, 0.0208333, -
0.0416667, -0.510417]
```

```
F3_TRI_P2 = [0.0729167, 0.354167, 0.0104167, 1.57292, 0.0104167, -0.177083, 0.104167, -
0.0833333, -0.864583]
        [0.0, -0.0625, 0.0, -0.125, -0.125, 0.0625, -0.0625, 1.25, 0.0625]
        [0.0104167, 0.354167, 0.0729167, 0.0104167, 1.57292, -0.864583, 0.104167, -
0.0833333, -0.177083]
        [0.125, 0.28125, -0.03125, 0.90625, -0.375, 0.15625, 0.59375, 0.28125, -0.9375]
        [-0.03125, 0.28125, 0.125, -0.375, 0.90625, -0.9375, 0.59375, 0.28125, 0.15625]
        [0.0208333, 0.177083, 0.0520833, -0.260417, 0.645833, -0.510417, 0.114583,
0.802083, -0.0416667]
        [-0.114583, -0.583333, -0.114583, -0.427083, -0.427083, 0.822917, 1.91667, -
0.895833, 0.822917]
        [0.0208333, 0.0208333, 0.0208333, -0.104167, -0.104167, -0.0416667, 0.895833,
0.333333, -0.0416667]
        [0.0520833, 0.177083, 0.0208333, 0.645833, -0.260417, -0.0416667, 0.114583,
0.802083, -0.510417]
F3_TRI_P3 = [0.0208333, -0.0416667, -0.0416667, -0.0416667, 0.895833, 0.145833, -
0.0416667, 0.145833, -0.0416667]
        [-0.0729167, -0.291667, -0.0729167, -0.260417, -0.260417, 0.489583, -0.166667,
1.14583, 0.489583]
        [-0.0416667, -0.0416667, 0.0208333, 0.895833, -0.0416667, -0.0416667, -
0.0416667, 0.145833, 0.145833]
        [-0.0729167, -0.260417, -0.166667, -0.291667, -0.260417, 1.14583, -0.0729167,
0.489583, 0.489583]
        [-0.166667, -0.260417, -0.0729167, -0.260417, -0.291667, 0.489583, -0.0729167,
0.489583, 1.14583]
        [-0.135417, -0.322917, -0.0416667, -0.0416667, -0.322917, 0.395833, -0.135417,
0.802083, 0.802083]
       [-0.0416667, 0.895833, -0.0416667, -0.0416667, -0.0416667, 0.145833, 0.0208333,
-0.0416667, 0.145833]
        [-0.135417, -0.0416667, -0.135417, -0.322917, -0.322917, 0.802083, -0.0416667,
0.395833, 0.802083]
        [-0.0416667, -0.322917, -0.135417, -0.322917, -0.0416667, 0.802083, -0.135417,
0.802083, 0.395833]
F3_TRI_L0 = [0.351209, 0.158501, 0.0335005, 0.158501, 0.00745885, 0.0855838, 0.0335005,
0.0855838, 0.132459]
       [-0.0572595, 0.138053, 0.171907, -0.0416345, 0.0416988, 0.0573238, -0.00778035,
0.0338863, 0.0156572]
        [0.00662294, -0.0142104, 0.00141461, 0.0118313, -0.0142104, -0.00900206,
0.00662294, -0.00379372, -0.00379372]
       [-0.0572595, -0.0416345, -0.00778035, 0.138053, 0.0416988, 0.0338863, 0.171907,
0.0573238, 0.0156572]
        [0.00778035, 0.00517618, -3.21502e-05, 0.00517618, -0.00263632, -0.0104488, -
3.21502e-05, -0.0104488, -0.013053]
       [0.0310249, -0.0783501, 0.00237912, 0.0127958, 0.0258166, 0.0127958, -0.0106417,
-0.0314751, -0.0106417]
        [0.00662294, 0.0118313, 0.00662294, -0.0142104, -0.0142104, -0.00379372,
0.00141461, -0.00900206, -0.00379372]
        [0.0310249, 0.0127958, -0.0106417, -0.0783501, 0.0258166, -0.0314751,
0.00237912, 0.0127958, -0.0106417]
        [-0.0697659, 0.0578382, 0.0526299, 0.0578382, 0.138567, 0.11513, 0.0526299,
0.11513, 0.128151]
F3_TRI_L1 = [0.00141461, -0.0142104, 0.00662294, -0.0142104, 0.0118313, -0.00379372,
0.00662294, -0.00379372, -0.00900206]
       [0.171907, 0.138053, -0.0572595, 0.0416988, -0.0416345, 0.0156572, -0.00778035,
0.0338863, 0.0573238]
        [0.0335005, 0.158501, 0.351209, 0.00745885, 0.158501, 0.132459, 0.0335005,
0.0855838, 0.0855838]
        [-3.21502e-05, 0.00517618, 0.00778035, -0.00263632, 0.00517618, -0.013053, -
3.21502e-05, -0.0104488, -0.0104488]
```

```
       [-0.00778035, -0.0416345, -0.0572595, 0.0416988, 0.138053, 0.0156572, 0.171907,
0.0573238, 0.0338863]
        [0.0526299, 0.0578382, -0.0697659, 0.138567, 0.0578382, 0.128151, 0.0526299,
0.11513, 0.11513]
        [0.00662294, 0.0118313, 0.00662294, -0.0142104, -0.0142104, -0.00379372,
0.00141461, -0.00900206, -0.00379372]
        [-0.0106417, 0.0127958, 0.0310249, 0.0258166, -0.0783501, -0.0106417,
0.00237912, 0.0127958, -0.0314751]
        [0.00237912, -0.0783501, 0.0310249, 0.0258166, 0.0127958, -0.0106417, -
0.0106417, -0.0314751, 0.0127958]
F3_TRI_L2 = [0.00141461, -0.0142104, 0.00662294, -0.0142104, 0.0118313, -0.00379372,
0.00662294, -0.00379372, -0.00900206]
        [-3.21502e-05, -0.00263632, -3.21502e-05, 0.00517618, 0.00517618, -0.0104488,
0.00778035, -0.013053, -0.0104488]
        [0.00662294, -0.0142104, 0.00141461, 0.0118313, -0.0142104, -0.00900206,
0.00662294, -0.00379372, -0.00379372]
       [0.171907, 0.0416988, -0.00778035, 0.138053, -0.0416345, 0.0338863, -0.0572595,
0.0156572, 0.0573238]
       [-0.00778035, 0.0416988, 0.171907, -0.0416345, 0.138053, 0.0573238, -0.0572595,
0.0156572, 0.0338863]
      [-0.0106417, 0.0258166, 0.00237912, 0.0127958, -0.0783501, 0.0127958, 0.0310249,
-0.0106417, -0.0314751]
        [0.0335005, 0.00745885, 0.0335005, 0.158501, 0.158501, 0.0855838, 0.351209,
0.132459, 0.0855838]
        [0.0526299, 0.138567, 0.0526299, 0.0578382, 0.0578382, 0.11513, -0.0697659,
0.128151, 0.11513]
        [0.00237912, 0.0258166, -0.0106417, -0.0783501, 0.0127958, -0.0314751,
0.0310249, -0.0106417, 0.0127958]
F3_TRI_L3 = [0.000128601, -0.0154964, 0.0122814, -0.0154964, 0.0244342, -0.0120242,
0.0122814, -0.0120242, -0.00334362]
        [-0.00176826, -0.00610854, -0.00176826, 0.0416345, 0.0416345, 0.0225373,
0.143197, 0.0529192, 0.0225373]
        [0.0122814, -0.0154964, 0.000128601, 0.0244342, -0.0154964, -0.00334362,
0.0122814, -0.0120242, -0.0120242]
        [-0.00176826, 0.0416345, 0.143197, -0.00610854, 0.0416345, 0.0529192, -
0.00176826, 0.0225373, 0.0225373]
        [0.143197, 0.0416345, -0.00176826, 0.0416345, -0.00610854, 0.0225373, -
0.00176826, 0.0225373, 0.0529192]
        [0.0502508, 0.0242091, -0.0148534, 0.13098, 0.0242091, 0.0372299, 0.0502508,
0.0710841, 0.0710841]
        [0.0122814, 0.0244342, 0.0122814, -0.0154964, -0.0154964, -0.0120242,
0.000128601, -0.00334362, -0.0120242]
        [0.0502508, 0.13098, 0.0502508, 0.0242091, 0.0242091, 0.0710841, -0.0148534,
0.0372299, 0.0710841]
        [-0.0148534, 0.0242091, 0.0502508, 0.0242091, 0.13098, 0.0710841, 0.0502508,
0.0710841, 0.0372299]

// === Triangular Transfer Matrices ========================================

M2_TRI_R = [1.66667, -0.333333, -0.333333]
        [-0.333333, -0.333333, 1.66667]
        [-0.333333, 1.66667, -0.333333]
M2_TRI_S = [0.222222, 0.0555556, 0.0555556]
        [0.0555556, 0.0555556, 0.222222]
        [0.0555556, 0.222222, 0.0555556]

M3_TRI_R = [-0.9, 1.93751, 0.106559, 0.106559, 0.362493, -0.306559, -0.306559]
        [-0.9, 0.106559, 0.106559, 1.93751, -0.306559, -0.306559, 0.362493]
        [-0.9, 0.106559, 1.93751, 0.106559, -0.306559, 0.362493, -0.306559]
```

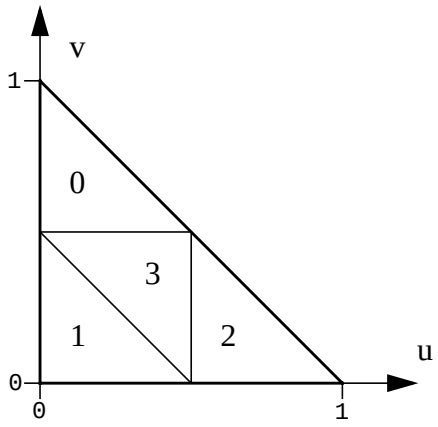
```
        [1.35, -0.808915, 0.22284, -0.808915, -0.641085, 2.32716, -0.641085]
        [1.35, 0.22284, -0.808915, -0.808915, 2.32716, -0.641085, -0.641085]
        [1.35, -0.808915, -0.808915, 0.22284, -0.641085, -0.641085, 2.32716]
M3_TRI_S = [0.025, 0.025, 0.025, 0.05, 0.05, 0.05]
        [0.0800834, 0.001292, 0.001292, 0.0203439, 0.00258401, 0.0203439]
        [0.001292, 0.001292, 0.0800834, 0.00258401, 0.0203439, 0.0203439]
        [0.001292, 0.0800834, 0.001292, 0.0203439, 0.0203439, 0.00258401]
        [0.000472116, 0.0292636, 0.0292636, 0.00743392, 0.0585271, 0.00743392]
        [0.0292636, 0.0292636, 0.000472116, 0.0585271, 0.00743392, 0.00743392]
        [0.0292636, 0.000472116, 0.0292636, 0.00743392, 0.00743392, 0.0585271]

M2_TRI_U = [0.166667, 0.166667]
        [0.166667, 0.666667]
        [0.666667, 0.166667]
M3_TRI_U = [0.333333, 0.333333]
        [0.101287, 0.101287]
        [0.101287, 0.797427]
        [0.797427, 0.101287]
        [0.470142, 0.470142]
        [0.470142, 0.0597159]
        [0.0597159, 0.470142]
```

Using the Coefficients

We start with triangular bases, which are actually a little easier to understand, because of their non-tensor nature. If you are implementing wavelets over geometry, rather than (say) a 2D image, you can probably ignore the quadrilateral wavelets altogether.

Example 1: M2 Basis over a Triangle.

Parameterization



$$\mathbf{B} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$$f(u,v) = av + b(1-u-v) + cu$$

Push:

```
child[0].B = M2_TRI_P0 * B
```

```
child[1].B = M2_TRI_P1 * B
child[2].B = M2_TRI_P2 * B
child[3].B = M2_TRI_P3 * B
```

Pull:

```
B  = M2_TRI_L0 * child[0].B
B += M2_TRI_L1 * child[1].B
B += M2_TRI_L2 * child[2].B
B += M2_TRI_L3 * child[3].B
```

FindTransferCoeffs

```
K[i][j] = KernelSample(SourceMap(M2_TRI_U[i]), DestMap(M2_TRI_U[j]))
T = R * K * S
```

- SourceMap(uv) should map the 2D U parameter to the corresponding point and normal on the source patch. DestMap(uv) should do the same for the destination patch.
- KernelSample(x, nx, y, ny) should return the transfer function sampled between x and y, with corresponding normals nx and ny. For radiosity, this function is as in **(6)**.

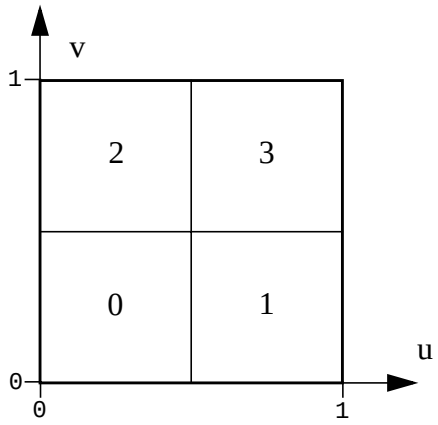
DoTransfer

```
dest.B += T * src.B
```

Example 2: M2 Tensor-Product Basis over a Quadrilateral

Parameterization

The 2D quadrilateral function basis is formed from the tensor product of two 1D line function bases:



$$B = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$f(u, v) = a + bu + cv + duv$$

Push:

```
child[0].B = M2_LINE_P0 * B * trans(M2_LINE_P0);
```

```
child[1].B = M2_LINE_P0 * B * trans(M2_LINE_P1);
child[2].B = M2_LINE_P1 * B * trans(M2_LINE_P0);
child[3].B = M2_LINE_P1 * B * trans(M2_LINE_P1);
```

Pull

```
B  = M2_LINE_L0 * child[0].B * trans(M2_LINE_L0);
B += M2_LINE_L0 * child[1].B * trans(M2_LINE_L1);
B += M2_LINE_L1 * child[1].B * trans(M2_LINE_L0);
B += M2_LINE_L1 * child[1].B * trans(M2_LINE_L1);
```

FindTransferCoeffs

```
K[i][j] = KernelSample(SourceMap(M2_QUAD_U[i]), DestMap(M2_QUAD_U[j]))
T = LeftTSPMult(M2_LINE_R, RightTSPMult(K, M2_LINE_S));
```

This requires the two routines:

```
LeftTSPMult(A, B)
     Int       n = A.Rows();
     Int       i, j, k, l, s, t, m;
     Matrix    P, result;

     result = 0;

     for (m = 0; m < B.Cols(); m++)
          P = 0;
          for (j = 0; j < n; j++)
               for (k = 0, t = 0; k < n; k++)
                    for (l = 0; l < n; l++, t++)
                         P[j][k] += A[j][l] * B[t][m];

          for (i = 0, s = 0; i < n; i++)
               for (j = 0; j < n; j++, s++)
                    for (k = 0; k < n; k++)
                         result[s][m] += A[i][k] * P[j][k];

RightTSPMult(A, B)
     Int       n = B.Rows();
     Int       i, j, k, l, s, t, m;
     Matrix    P, result;

     result = 0;

     for (m = 0; m < A.Rows(); m++)
          P = 0;
          for (l = 0; l < n; l++)
               for (i = 0, s = 0; i < n; i++)
                    for (j = 0; j < n; j++, s++)
                         P[i][l] += B[j][l] * A[m][s];

          for (k = 0, t = 0; k < n; k++)
```

```
for (l = 0; l < n; l++, t++)
     for (i = 0; i < n; i++)
          result[m][t] += B[i][k] * P[i][l];
```

DoTransfer

```
dest.Bc += T * src.Bc
```

where Bc is B rewritten as the column vector, $\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$.

Notes

- See Tensor.cc for the definitions of LeftTSPMult and RightTSPMult.
- The various _LINE_ matrices are 2x2; K and T are 4x4.
- The coefficients generated by MTE by default are over the interval [0, 1]. Most of the original wavelet papers use the interval [-1, 1], which is why the coefficients they give may not match the ones here.
- It might seem less complicated just to generate _QUAD_ versions for P_i, L_i, R and S; we could then follow exactly the same procedures as in the M2 triangle example. Unfortunately, this would make the calculations slower; the redundancy inherent in the tensor-product bases means it is faster to use specialised routines that operate directly with the 1D versions of the matrices.
- The tensor push/pull operations shown should be optimised with common sub-expression elimination for maximum efficiency.

Flatlets

For Flatlet bases, as noted before, R = S = I, so T = K. The kernel samples are just the formfactors between the panes of the source and destination patches.