The part of the wavelet transformation process that was modified to facilitate multithreading is highlighted

// Image Compression Toolbox v1.2
// written by
// Satish Kumar S
// satishkumr@lycos.com
//
// Copyright 1999 Satish Kumar S
//
// Permission is granted to use this software for research purposes as
// long as this notice stays attached to this software.
//
// The wavelet class is modified by Hadassa Baker to allow
// multithreaded execution

#ifndef _WAVELET_H
#define _WAVELET_H
#include "Monitor.h"
#include <iostream>

using namespace std;

class Filter
{
public:
   double *coeffs; // filter coefficients.
   int size; // length of the filter.
   int startIndex; // the starting index of the coefficients.

   Filter(int size, int startIndex, double *coeffs = NULL);
};

// The wavelet transformation class inherits the ZThread runnable class.
// The Runnable class provides a common interface to execute any arbitrary task

class Wavelet : public Runnable {
   CountedPtr<Monitor> monitor;
   CountedPtr<Monitor2> monitor2;
   CountedPtr<Image> image;

   Filter *anaLow, *anaHigh; // low and high pass filters for analysis.
   Filter *synLow, *synHigh; // low and high pass filters for synthesis.
   int nsteps;

   bool Encoding;
   bool waitingForCancel;

private:
   void PeriodicExtension(double *output, int size, int npad);
   void SymmetricExtension(double *output, int size, int left_ext, int right_ext,
      int symmetry, int npad);
   void TransformStep(double *data, int datasize, int npad, Filter *lpf,
      Filter *hpf, double *dest);
void InverseStep(double *data, int datasize, int npad, Filter *lpf, Filter *hpf, double *dest);
int Transform();
int Inverse();

public:
    Wavelet(bool encode,int bands, CountedPtr<Monitor>& m, CountedPtr<Monitor2>& m2, CountedPtr<Image>& img);

    void run()
    {
        //cout << "WT Beginning" << endl;
        if (Encoding)
        {
            Transform();
            waitingForCancel = true;
            //printf(" after transform \n");
            while(!monitor->isCanceled()) // Hold here until transformation is complete...
                Thread::sleep(100);
            monitor2->SBStart(); // Start subband initialization
        }
        else
        {
            monitor2->WaitToStartWT(); //wait to start inverse transformation
            Inverse();
            waitingForCancel = true;
            //printf(" after transform \n");
            while(!monitor->isCanceled()) // Hold here until transformation is complete...
                Thread::sleep(100);
            monitor2->DelMemStart(); // Start memory free
        }
    }
};

#endif