Skype Video Streaming Test Task - Implement a Jitterbuffer

Description

The jitterbuffer's task is to receive packets from the network, each packet containing one fragment of an encoded video frame, and to then:

- Reassemble these packets into frames - there may be 1-n packets (fragments) in every frame, and they can be reassembled by simply combining them together in the right order – any extra header for the packets will have already been stripped.
- Pass these assembled frames to the decoder to decode. Note the decoder can only decode the frames in the correct order, and with no gaps (frames skipped) - you are guaranteed that every fragment of every frame will arrive eventually, but may have to wait when some fragments are late.
- Send these decoded frames to the renderer to display.

Packets may be delayed, and even arrive out of order, and it is up to the jitterbuffer to wait until it has all the packets for the next frame, assemble them into the frame, and then pass to the decoder.

Because some packets may arrive late, you can have situations where you are stuck waiting for some late packet and thus unable to render anything for say one second, and then suddenly have many frames available to decode + render. In such a case, you will want to try to 'catch up' (decode+render the queued frames quickly), but at a reasonable pace, so it is up to the jitterbuffer to control this to try to make what is seen appear to the user as smoothly as possible. If you want to get fancy, you can try to make the jitterbuffer adapt the delay to conditions, for example in a situation where packets are often arriving late, you may want to generally wait longer before decoding frames and passing them to the renderer even when they are available, so that when one frame is suddenly late, it's easier to smooth over.

You should decide whether the jitterbuffer should create its own thread from which to do the decoding and rendering, or should just be driven solely by the thread that is passing it the fragments.

You should provide some form of unit-tests to you used to prove that your solution works - this doesn't need to work with real video data + decoding + rendering of course.

And you should provide some documentation about how your solution works, why you solved it in the way you did, and what limitations/issues you see in your solution (and how things could be improved). Indicate if you identified any trade-offs to be made during implementation, and why you decided to resolve them as you did.

Also, keep track of the time spent on this task, and report how long you spent in doing the different things. Try to keep the total time you spend on this task to no more than 20 hours, and to come up with the best solution that is practical within this timeframe.

Please ask us if you have any questions about the specification or requirements.

Interfaces

```
class IDecoder
{
public:

   /*
    Returns the size of the data written to the outputBuffer, will be no more than 1mb.
    */
   virtual int DecodeFrame(const char* buffer, int length, char* outputBuffer) = 0;

   ~IDecoder() {}
};

class IRenderer
{
public:

   /*
    Renders the given buffer. This call will not block for any significant period, and
    the buffer will be copied internally so can be deleted/reused as soon as this call
    is completed.
    */
   virtual void RenderFrame(const char* buffer, int length) = 0;

   ~IRenderer() {}
};


/*
 This is the class that you need to implement!
 */
class IJitterBuffer
{
public:

   IJitterBuffer(IDecoder* decoder, IRenderer* renderer);

   /*
    Should copy the given buffer, as it may be deleted/reused immediately following this call.

    @param frameNumber - will start at zero for the call
    @param fragmentNumber – specifies what position this fragment is within the given
       frame – the first fragment number in each frame is number zero
    @param numFragmentsInThisFrame - is guaranteed to be identical for all fragments
       with the same frameNumber
    */
   virtual void ReceivePacket(
      const char* buffer,
      int length,
      int frameNumber,
      int fragmentNumber,
      int numFragmentsInThisFrame) = 0;

   ~IJitterBuffer() {}
};
```