Ok, so first things first - cubic spline interpolation. You're probably fine with this bit, but I want to write it out to solidify the notation system I'm using, otherwise it could be confusing.

- We have a set of points (x_i, y_i) such that

$$S_i(x_i) = y_i, i = 0, ..., n \tag{1}$$

  S is our set of unknown spline functions - we know the set of $n + 1$ fixed points they must pass through, but not what the functions are.

- Since we have $n + 1$ points, we have n intervals, so n spline functions. For each cubic spline, we will have four unknown coefficients, so in total we get $4n$ unknown coefficients.

- We can set a certain number of conditions on our problem, to help eliminate some of the unknowns. For example, with natural splines, we know the second derivatives at the very end points must be zero, ie

$$S''(x_0) = S''(x_n) = 0 \tag{2}$$

- And we also know, because each spline needs to join with those preceding and following it, that

$$S''_{i-1}(x_i) = S''_i(x_i) \tag{3}$$

  (ie the function is not discontinuous in the second derivative at any point).

- So we can use these facts to set up a system of linear equations, which can be easily represented in matrix form - this is where your tri-diagonal matrix comes in.

We can write the system as

$$T\mathbf{d} = \mathbf{m} \tag{4}$$

where T is the tri-diagonal matrix which I can't be bothered writing out, but consists of known integers ie see the bottom of this page http://mathworld.wolfram.com/CubicSpline.html . $\mathbf{m}$ is a vector of knowns which is derived from the fixed points y_i, and $\mathbf{d}$ is our vector of unknowns, which corresponds to the values of the first derivatives of the spline at the junction points between the splines (so $\mathbf{d}$ is not actually the solution to the

problem, but it is very close to it - once you have **d** then getting the spline coefficients is trivial). This notation clashes a bit with the page you linked me to, sorry about that. Hopefully it's clear.

This is a simple linear system, which can easily be solved using the standard gaussian elimination procedure, as I'm sure you know. The 'efficient' method described here is just an algebraic formalization of the division and subtraction procedures you would make normally - basically because the matrix is regular and predictable, we can create an algorithm for the division, subtraction and back-substitution. So it's actually exactly the same as a standard elimination method - you divide a row through by the first co-efficient so your first unknown has a unity coefficient, and then you select a suitable row to subtract from, etc., repeat. If you like I can go into more detail on traditional Gaussian elimination, this might make things clearer. The algorithm here is just filtering several elimination steps down into one, and it can do that because the matrix has some predictable properties.