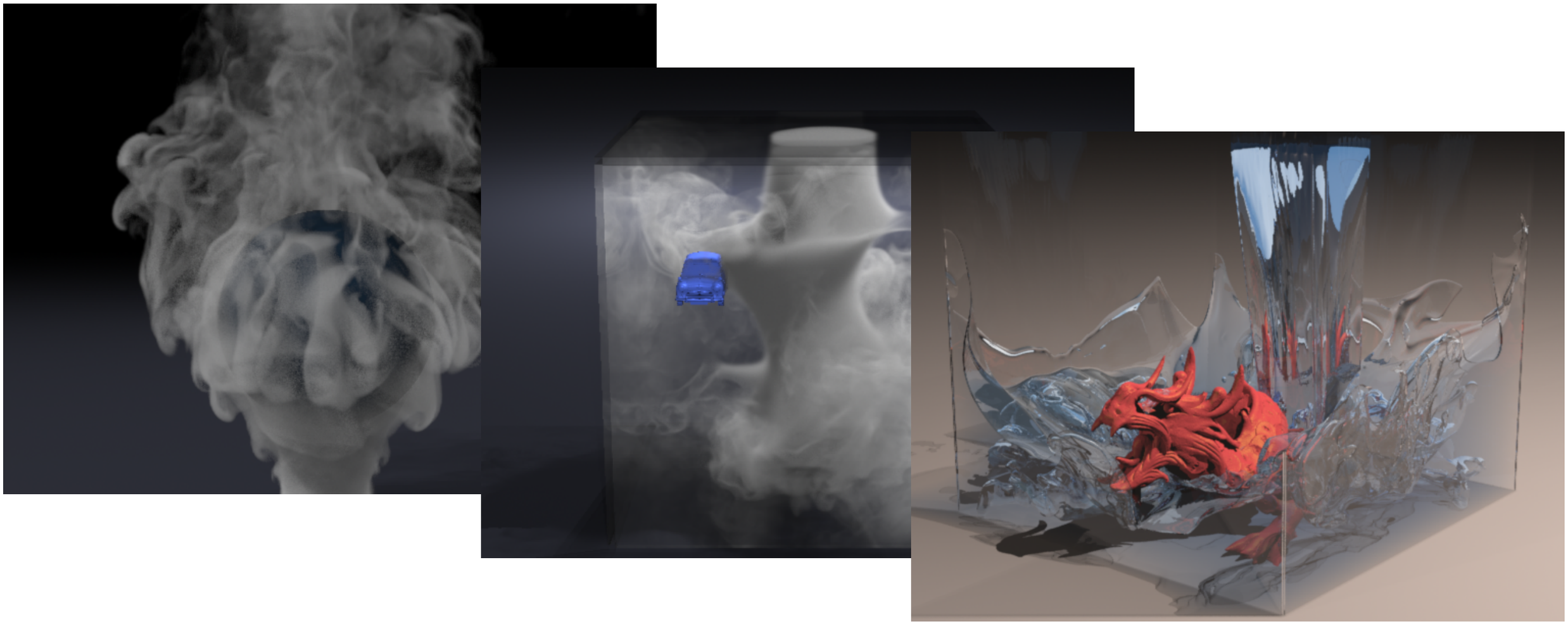

Sparse Matrix Computations

Storage formats and typical Use Scenarios

Recap

Example : The 3D Poisson equation

$$
\begin{pmatrix}
-6 & 1 & & & 1 & & & 1 & & & \\
1 & -6 & 1 & & & 1 & & & \ddots & & \\
 & 1 & -6 & 1 & & & \ddots & & & \ddots & \\
 & & \ddots & \ddots & \ddots & & & \ddots & & & 1 \\
1 & & & \ddots & \ddots & \ddots & & & \ddots & & \\
 & 1 & & & \ddots & \ddots & \ddots & & & \ddots & \\
 & & \ddots & & & \ddots & \ddots & \ddots & & & 1 \\
 & & & \ddots & & & \ddots & \ddots & \ddots & & \\
1 & & & & \ddots & & & \ddots & \ddots & \ddots & \\
 & \ddots & & & & \ddots & & & 1 & -6 & 1 \\
 & & \ddots & & & & 1 & & & 1 & -6 & 1 \\
 & & & 1 & & & & 1 & & & 1 & -6
\end{pmatrix}
\mathbf{x} = \mathbf{b}
$$

CSR Matrix structure (CSRMatrix.h)

LaplaceSolver_1_0

```
#pragma once

#include <memory>

struct CSRMatrix
{
    int mSize;
    std::unique_ptr<int> mRowOffsets;
    std::unique_ptr<int> mColumnIndices;
    std::unique_ptr<float> mValues;

    int* GetRowOffsets() { return mRowOffsets.get(); }
    int* GetColumnIndices() { return mColumnIndices.get(); }
    float* GetValues() { return mValues.get(); }
};
```

Matrix-Vector multiply (MatVecMultiply.cpp)

LaplaceSolver_1_0

```
#include "MatVecMultiply.h"

void MatVecMultiply(CSRMatrix& mat, const float *x, float *y)
{
    int N = mat.mSize;
    const auto rowOffsets = mat.GetRowOffsets();
    const auto columnIndices = mat.GetColumnIndices();
    const auto values = mat.GetValues();

#pragma omp parallel for
    for (int i = 0; i < N; i++)
    {
        y[i] = 0.;
        for (int k = rowOffsets[i]; k < rowOffsets[i+1]; k++) {
            const int j = columnIndices[k];
            y[i] += values[k] * x[j];
        }
    }
}
```

x[] and y[] are presumed to have arrays allocated on them, of size mat.mSize

CSR helper (CSRMatrixHelper.h)

LaplaceSolver_1_0

```
#include "CSRMatrix.h"

struct CSRMatrixHelper
{
    std::vector<std::map<int,float> > mSparseRows;

    CSRMatrixHelper(const int size) { mSparseRows.resize(size); }

    float& operator() (const int i, const int j)
    {
        if (i < 0 || i >= mSparseRows.size() || j < 0 || j >= mSparseRows.size())
            throw std::logic_error("Matrix index out of bounds");
        return mSparseRows[i].insert( {j, 0.} ).first->second;
    }

    CSRMatrix ConvertToCSRMatrix()
    {
       [ ... omitted ...]
    }
};
```

This helper assists us in constructing the matrix in a more intuitive way (i.e. by accessing/setting individual entries)

CSR helper (CSRMatrixHelper.h)

LaplaceSolver_1_0

```
struct CSRMatrixHelper
{
    std::vector<std::map<int,float> > mSparseRows;
    [ ... omitted ...]
    CSRMatrix ConvertToCSRMatrix()
    {
        int N = mSparseRows.size(); // Size of matrix
        int NNZ = 0; // Number of non-zero entries
        for (int i = 0; i < N; i++) NNZ += mSparseRows[i].size();

        CSRMatrix matrix { N }; // Initialize just matrix.mSize
        matrix.mRowOffsets.reset(new int [N + 1]); // Need a sentinel value in the end
        matrix.mColumnIndices.reset(new int [NNZ]);
        matrix.mValues.reset(new float [NNZ]);

        auto rowOffsets = matrix.GetRowOffsets();
        auto columnIndices = matrix.GetColumnIndices();
        auto values = matrix.GetValues();

        rowOffsets[0] = 0;
        for (int i = 0, k = 0; i < N; i++) {
            rowOffsets[i + 1] = rowOffsets[i] + mSparseRows[i].size(); // Mark where this row ends
            for (auto it = mSparseRows[i].begin(); it != mSparseRows[i].end(); it++) {
                columnIndices[k] = it->first;
                values[k] = it->second;
                k++;}}
       return matrix;
    }
};
```

New Laplacian (Laplacian.h)

LaplaceSolver_1_0

New Laplacian : Build and use CSR Matrix

```
#pragma once

#include "CSRMatrix.h"
#include "Parameters.h"

CSRMatrix BuildLaplacianMatrix();

void ComputeLaplacian(CSRMatrix& laplacianMatrix,
    const float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[XDIM][YDIM][ZDIM]);
```

New Laplacian (Laplacian.cpp)

LaplaceSolver_1_0

New Laplacian : Build and use CSR Matrix

```
#include "CSRMatrixHelper.h"
#include "Laplacian.h"
#include "MatVecMultiply.h"

inline int LinearIndex(const int i, const int j, const int k)
{ return ((i * YDIM) + j) * ZDIM + k; }

CSRMatrix BuildLaplacianMatrix() {
    static constexpr int matSize = XDIM * YDIM * ZDIM;
    CSRMatrixHelper matrixHelper(matSize);

    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++) {
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k) ) = -6.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i+1, j, k) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i-1, j, k) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j+1, k) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j-1, k) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k+1) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k-1) ) = 1.;}
    return matrixHelper.ConvertToCSRMatrix();
}

void ComputeLaplacian(CSRMatrix& laplacianMatrix,
    const float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[XDIM][YDIM][ZDIM]) {
    // Treat the arrays u & Lu as flattened vectors, and apply matrix-vector multiplication
    MatVecMultiply(laplacianMatrix, &u[0][0][0], &Lu[0][0][0]);
}
```

New CG routine (ConjugateGradients.cpp)

LaplaceSolver_1_0

```
void ConjugateGradients(
    CSRMatrix& matrix,
    float (&x)[XDIM][YDIM][ZDIM],
    const float (&f)[XDIM][YDIM][ZDIM],
    float (&p)[XDIM][YDIM][ZDIM],
    float (&r)[XDIM][YDIM][ZDIM],
    float (&z)[XDIM][YDIM][ZDIM],
    const bool writeIterations)
{
    // Algorithm : Line 2
    timerLaplacian.Restart(); ComputeLaplacian(matrix, x, z); timerLaplacian.Pause();
    Saxpy(z, f, r, -1);
    float nu = Norm(r);

    // Algorithm : Line 3
    if (nu < nuMax) return;

    // Algorithm : Line 4
    Copy(r, p);
    float rho=InnerProduct(p, r);

    // Beginning of loop from Line 5
    for(int k=0;;k++)
    {
        std::cout << "Residual norm (nu) after " << k << " iterations = " << nu << std::endl;

        // Algorithm : Line 6
        timerLaplacian.Restart(); ComputeLaplacian(matrix, p, z); timerLaplacian.Pause();
        float sigma=InnerProduct(p, z);
```

New main routine (main.cpp)

LaplaceSolver_1_0

```
[.. .]
Timer timerLaplacian;

int main(int argc, char *argv[])
{
    using array_t = float (&) [XDIM][YDIM][ZDIM];
    float *xRaw = new float [XDIM*YDIM*ZDIM];
    [.. .]
    array_t z = reinterpret_cast<array_t>(*zRaw);

    CSRMatrix matrix;

    // Initialization
    {
        Timer timer;
        timer.Start();
        InitializeProblem(x, f);
        matrix = BuildLaplacianMatrix(); // This takes a while ...
        timer.Stop("Initialization : ");
    }

    // Call Conjugate Gradients algorithm
    timerLaplacian.Reset();
    ConjugateGradients(matrix, x, f, p, r, z, false);
    timerLaplacian.Print("Total Laplacian Time : ");

    return 0;
}
```

New main routine (main.cpp)

LaplaceSolver_1_0

```
[.. .]
Timer timerLaplacian;

int main(int argc, char *argv[])
{
    using array_t = float (&) [XDIM][YDIM][ZDIM];
    float *xRaw = new float [XDIM*YDIM*ZDIM];
    [.. .]
    array_t z = reinterpret_cast<array_t>(*zRaw);

    CSRMatrix matrix;

    // Initialization
    {
        Timer timer;
        timer.Start();
        InitializeProblem(x, f);
        matrix = BuildLaplacianMatrix(); // This takes a while ...
        timer.Stop("Initialization : ");
    }

    // Call Conjugate Gradients algorithm
    timerLaplacian.Reset();
    ConjugateGradients(matrix, x, f, p, r, z, false);
    timerLaplacian.Print("Total Laplacian Time : ");

    return 0;
}
```

The explicitly constructed matrix now gets passed as an argument to the CG algorithm

How about symmetry?

Matrix has a symmetric structure

$$
\begin{pmatrix}
-6 & 1 & & & 1 & & & 1 & & & & \\
1 & -6 & 1 & & & 1 & & & \ddots & & & \\
 & 1 & -6 & 1 & & & \ddots & & & \ddots & & \\
 & & \ddots & \ddots & \ddots & & & \ddots & & & 1 & \\
1 & & & \ddots & \ddots & \ddots & & & \ddots & & & \\
 & 1 & & & \ddots & \ddots & \ddots & & & \ddots & & \\
 & & \ddots & & & \ddots & \ddots & \ddots & & & 1 & \\
 & & & \ddots & & & \ddots & \ddots & \ddots & & & 1 \\
1 & & & & \ddots & & & \ddots & \ddots & \ddots & & \\
 & \ddots & & & & \ddots & & & 1 & -6 & 1 & \\
 & & \ddots & & & & 1 & & & 1 & -6 & 1 \\
 & & & 1 & & & & 1 & & & 1 & -6
\end{pmatrix}
\mathbf{x} = \mathbf{b}
$$

How about symmetry?

(but not really - we have skipped rows)

$$
\begin{pmatrix}
-6 & 1 & & & 1 & & & 1 & & & & \\
1 & -6 & 1 & & & 1 & & & \ddots & & & \\
 & 1 & -6 & 1 & & & \ddots & & & \ddots & & \\
 & & \ddots & \ddots & \ddots & & & \ddots & & & 1 & \\
1 & & & \ddots & \ddots & \ddots & & & \ddots & & & \\
 & 1 & & & \ddots & \ddots & \ddots & & & \ddots & & \\
 & & \ddots & & & \ddots & \ddots & \ddots & & & 1 & \\
 & & & \ddots & & & \ddots & \ddots & \ddots & & & 1 \\
1 & & & & \ddots & & & \ddots & \ddots & \ddots & & \\
 & \ddots & & & & \ddots & & & 1 & -6 & 1 & \\
 & & \ddots & & & & 1 & & & 1 & -6 & 1 \\
 & & & 1 & & & & 1 & & & 1 & -6
\end{pmatrix}
\mathbf{x} = \mathbf{b}
$$

New Laplacian (Laplacian.cpp)

LaplaceSolver_1_0

```
#include "CSRMatrixHelper.h"
#include "Laplacian.h"
#include "MatVecMultiply.h"

inline int LinearIndex(const int i, const int j, const int k)
{ return ((i * YDIM) + j) * ZDIM + k; }

CSRMatrix BuildLaplacianMatrix() {
    static constexpr int matSize = XDIM * YDIM * ZDIM;
    CSRMatrixHelper matrixHelper(matSize);

    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++) {
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k) ) = -6.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i+1, j, k) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i-1, j, k) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j+1, k) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j-1, k) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k+1) ) = 1.;
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k-1) ) = 1.;}
    return matrixHelper.ConvertToCSRMatrix();
}

void ComputeLaplacian(CSRMatrix& laplacianMatrix,
    const float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[XDIM][YDIM][ZDIM]) {
    // Treat the arrays u & Lu as flattened vectors, and apply matrix-vector multiplication
    MatVecMultiply(laplacianMatrix, &u[0][0][0], &Lu[0][0][0]);
}
```

How about symmetry?

To restore symmetry - also skip columns

$$
\begin{pmatrix}
-6 & 1 & & & 1 & & & 1 & & & \\
1 & -6 & 1 & & & 1 & & & \ddots & & \\
 & 1 & -6 & 1 & & & \ddots & & & \ddots & \\
 & & \ddots & \ddots & \ddots & & & \ddots & & & 1 \\
1 & & & \ddots & \ddots & \ddots & & & \ddots & & \\
 & 1 & & & \ddots & \ddots & \ddots & & & \ddots & \\
 & & \ddots & & & \ddots & \ddots & \ddots & & & 1 \\
 & & & \ddots & & & \ddots & \ddots & \ddots & & \\
1 & & & & \ddots & & & \ddots & \ddots & \ddots & \\
 & \ddots & & & & \ddots & & & 1 & -6 & 1 \\
 & & \ddots & & & & 1 & & & 1 & -6 & 1 \\
 & & & 1 & & & & 1 & & & 1 & -6
\end{pmatrix}
\mathbf{x} = \mathbf{b}
$$

New Laplacian (Laplacian.cpp)

LaplaceSolver_1_1

```
[...]

CSRMatrix BuildLaplacianMatrixNoBoundary()
{
    static constexpr int matSize = XDIM * YDIM * ZDIM;
    CSRMatrixHelper matrixHelper(matSize);

    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++) {
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k) ) = -6.;
        if (i < XDIM-2)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i+1, j, k) ) = 1.;
        if (i > 1)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i-1, j, k) ) = 1.;
        if (j < YDIM-2)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j+1, k) ) = 1.;
        if (j > 1)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j-1, k) ) = 1.;
        if (k < ZDIM-2)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k+1) ) = 1.;
        if (k > 1)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k-1) ) = 1.;
    }

    return matrixHelper.ConvertToCSRMatrix();
}
[.. .]
```

New Laplacian (Laplacian.cpp)

LaplaceSolver_1_1

```
[...]

CSRMatrix BuildLaplacianMatrixNoBoundary()
{
    static constexpr int matSize = XDIM * YDIM * ZDIM;
    CSRMatrixHelper matrixHelper(matSize);

    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++) {
        matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k) ) = -6.;
        if (i < XDIM-2)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i+1, j, k) ) = 1.;
        if (i > 1)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i-1, j, k) ) = 1.;
        if (j < YDIM-2)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j+1, k) ) = 1.;
        if (j > 1)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j-1, k) ) = 1.;
        if (k < ZDIM-2)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k+1) ) = 1.;
        if (k > 1)
            matrixHelper( LinearIndex(i, j, k), LinearIndex(i, j, k-1) ) = 1.;
    }

    return matrixHelper.ConvertToCSRMatrix();
}
[.. .]
```

New main routine (main.cpp)

LaplaceSolver_1_1

```
[.. .]
int main(int argc, char *argv[])
{
    using array_t = float (&) [XDIM][YDIM][ZDIM];
    float *xRaw = new float [XDIM*YDIM*ZDIM];
    [.. .]
    array_t z = reinterpret_cast<array_t>(*zRaw);

    CSRMatrix matrix;

    // Initialization
    {
        Timer timer;
        timer.Start();
        InitializeProblem(x, f);
        matrix1 = BuildLaplacianMatrix(); // This takes a while ...
        matrix2 = BuildLaplacianMatrixNoBoundary(); // This takes a while ...
        timer.Stop("Initialization : ");
    }

    // Call Conjugate Gradients algorithm
    timerLaplacian.Reset();
    ConjugateGradients(matrix1, matrix2, x, f, p, r, z, false);
    timerLaplacian.Print("Total Laplacian Time : ");}
[.. .]
}
```

New CG routine (ConjugateGradients.cpp)

LaplaceSolver_1_1

```
void ConjugateGradients(
    CSRMatrix& matrix1, CSRMatrix& matrix2,
    float (&x)[XDIM][YDIM][ZDIM],
    const float (&f)[XDIM][YDIM][ZDIM],
    float (&p)[XDIM][YDIM][ZDIM],
    float (&r)[XDIM][YDIM][ZDIM],
    float (&z)[XDIM][YDIM][ZDIM],
    const bool writeIterations)
{
    // Algorithm : Line 2
    timerLaplacian.Restart(); ComputeLaplacian(matrix1, x, z); timerLaplacian.Pause();
    Saxpy(z, f, r, -1);
    float nu = Norm(r);

    // Algorithm : Line 3
    if (nu < nuMax) return;

    // Algorithm : Line 4
    Copy(r, p);
    float rho=InnerProduct(p, r);

    // Beginning of loop from Line 5
    for(int k=0;;k++)
    {
        std::cout << "Residual norm (nu) after " << k << " iterations = " << nu << std::endl;

        // Algorithm : Line 6
        timerLaplacian.Restart(); ComputeLaplacian(matrix2, p, z, true); timerLaplacian.Pause();
        float sigma=InnerProduct(p, z);
```

New Laplacian (Laplacian.cpp)

LaplaceSolver_1_1

```
[...]

void ComputeLaplacian(CSRMatrix& laplacianMatrix,
    const float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[XDIM][YDIM][ZDIM], bool usingTranspose)
{
    // Treat the arrays u & Lu as flattened vectors, and apply matrix-vector multiplication
    if(usingTranspose)
        MatTransposeVecMultiply(laplacianMatrix, &u[0][0][0], &Lu[0][0][0]);
    else
        MatVecMultiply(laplacianMatrix, &u[0][0][0], &Lu[0][0][0]);
}
```

Matrix-Vector multiply (MatVecMultiply.cpp)

LaplaceSolver_1_1

```
#include "MatVecMultiply.h"

void MatVecMultiply(CSRMatrix& mat, const float *x, float *y)
{
    [.. .]
}

void MatTransposeVecMultiply(CSRMatrix& mat, const float *x, float *y)
{
    int N = mat.mSize;
    const auto rowOffsets = mat.GetRowOffsets();
    const auto columnIndices = mat.GetColumnIndices();
    const auto values = mat.GetValues();

    for (int i = 0; i < N; i++) y[i] = 0.;

    // What about OpenMP ?
    for (int i = 0; i < N; i++)
        for (int k = rowOffsets[i]; k < rowOffsets[i+1]; k++) {
            const int j = columnIndices[k];
            y[j] += values[k] * x[i];
        }
}
```

Matrix-Vector multiply (MatVecMultiply.cpp)

LaplaceSolver_1_2

```
[.. .]

void SymmetricLowerTriangularMatVecMultiply(CSRMatrix& mat, const float *x, float *y)
{
    int N = mat.mSize;
    const auto rowOffsets = mat.GetRowOffsets();
    const auto columnIndices = mat.GetColumnIndices();
    const auto values = mat.GetValues();

#pragma omp parallel for
    for (int i = 0; i < N; i++)
    {
        y[i] = 0.;
        for (int k = rowOffsets[i]; k < rowOffsets[i+1]; k++) {
            const int j = columnIndices[k];
            y[i] += values[k] * x[j];
        }
    }

    // What about OpenMP ?
    for (int i = 0; i < N; i++)
        for (int k = rowOffsets[i]; k < rowOffsets[i+1] - 1; k++) { // Note the "-1"
            const int j = columnIndices[k];                         // We are skipping the diagona
entry
            y[j] += values[k] * x[i];
        }
}
```