The Finite Element Method by Example in Qt/C++

Krzysztof Napiontek
knapiontek@gmail.com

Version 1.0
September 12, 2015

Abstract

The aim of this article is help with understanding The Finite Element Method in mechanics. The theoretical part is reduced to its minimum and most of the work is focused on diagrams, graphics and examples. The relevant part of source code calculating tetrahedron truss is presented at the end of each section along with its visualized output.

Contents

1 Introduction **2**

2 The Concrete Example **2**
2.1 Global Data Definition in the Example . . . 2

3 The Finite Element Method **3**
3.1 Tetrahedron Truss Breakdown . . . 3
3.2 The Stiffness Matrix of an Element in Local Coordinates . . . 4
3.3 Coordinate Transformation . . . 5
3.3.1 Global and Local Coordinates Relations . . . 5
3.3.2 The Point Displacement Relations . . . 5
3.3.3 Global and Local Forces Relations . . . 6
3.4 The Stiffness Matrix of an Element in Global Coordinates . . . 6
3.5 Global Stiffness Matrix Aggregation . . . 7
3.6 Fixing a Point Displacement . . . 8
3.7 Implementation . . . 9

4 Solving Linear Equations by LU Decomposition Method **11**
4.1 Explanation . . . 11
4.2 Solving an Example . . . 12
4.3 Pitfalls of Direct Methods . . . 13
4.4 The Concrete Example Result Presentation . . . 13
4.5 Implementation of LU Decomposition Method . . . 13

5 The 3D Data Presentation in the Documentation **15**
5.1 Point Rotation . . . 15
5.2 An Example of Rotated Tetrahedron . . . 16
5.3 Implementation of the Rotation Procedure . . . 16

6 Summary **16**

1 Introduction

There are many works presenting FEM, most of them very complex and comprehensive with purpose of full explanation of all details of the method. A reader who needs explanation of general idea behind finds it difficult to understand. This work tries to extract the most important aspects in a way helping to understand all knowledge almost at the first read. The relevant source code added at the end of each section calculates and visualizes the simplest 3D truss object - tetrahedron. The Hook's Law is the only equation borrowed from mechanics. There is also a need to understand trigonometry and basic matrix operations before reading.

2 The Concrete Example

For the purpose of this article the complete concrete example was implemented. It includes data type definitions, construction of a matrix equation, its solution and in final the presentation in graphics format. A number of dumps of the equation data have been generated for better understanding parts of implementation. The source code presented in this work was tested with Qt 5.1 in Windows environment. It does not use any platform specific code and can be easily ported to other systems.

The final result of the example is presented on the figure 1. It contains an original geometry of the truss, also geometry deformation by an external force and reactions in fixed points. Detailed explanation will be given in following chapters.

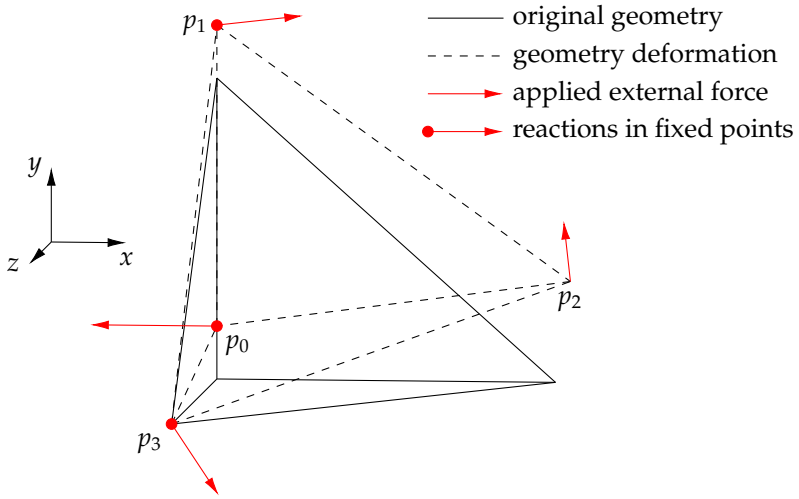

Figure 1: The Graphics Generated by the Example

2.1 Global Data Definition in the Example

```
struct Point3D
{
  double x, y, z;
};

// indices of first and second end of an element
struct Element
{
  int p1, p2;
};

const int point_no = 4;
const int element_no = 6;

// geometry nodes
```

```
const Point3D point_list[point_no] = {
  {-2.0,-2.0,-2.0 }, // 0
  {-2.0, 2.0,-2.0 }, // 1
  { 2.5,-2.0,-2.0 }, // 2
  {-2.0,-2.0, 4.0 } // 3
};

// geometry, refers to point_list or/and output_list
const Element element_list[element_no] = {
  { 0, 1 }, // 0
  { 1, 2 }, // 1
  { 2, 0 }, // 2
  { 3, 0 }, // 3
  { 3, 1 }, // 4
  { 3, 2 } // 5
};

// fixed point displacement list, boolean values
Point3D fix_list[point_no] = {
  { 1, 0, 1 }, // 0
  { 1, 0, 1 }, // 1
  { 0, 0, 0 }, // 2
  { 1, 1, 1 } // 3
};

// forces attached to coresponding points
Point3D force_list[point_no] = {
  { 0.0, 0.0, 0.0 }, // 0
  { 0.0, 0.0, 0.0 }, // 1
  { 0.0,30.0,30.0 }, // 2
  { 0.0, 0.0, 0.0 } // 3
};

// displaced points
Point3D output_list[point_no] = {
  { 0.0, 0.0, 0.0 }, // 0
  { 0.0, 0.0, 0.0 }, // 1
  { 0.0, 0.0, 0.0 }, // 2
  { 0.0, 0.0, 0.0 } // 3
};
```

3 The Finite Element Method

The FEM analyse in this chapter finds relation between forces and displacements in global coordinates. At first it breaks down the truss into separate elements, finds relation of them in local coordinates using the Hook's Law and in next step it transforms it to global coordinate system for each element. Finally it aggregates all elements in one global matrix of stiffness. The lower case bold font is reserved for local and the upper case bold font for global matrices and vectors.

3.1 Tetrahedron Truss Breakdown

The first step of analyses requires the truss breakdown. Each element will be analysed separately and finally all of them will be aggregated in global matrix. Separation of elements is shown on the figure 2.

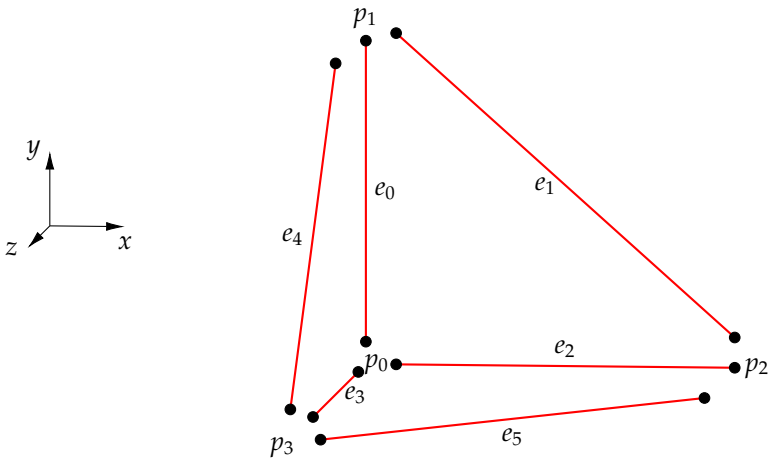

Figure 2: Truss Breakdown

3.2 The Stiffness Matrix of an Element in Local Coordinates

Relation between axial forces f_0, f_1 and axial displacements $\Delta p_0, \Delta p_1$ in local coordinates in respect of the Hook's Law as depicted on the figure 3 is described by equations 1 and 2 . Despite of 3D analyses most of the figures use 2D coordinates for simplicity.

$$f_0 = \frac{EA}{L}(\Delta p_0 - \Delta p_1) \tag{1}$$

$$f_1 = \frac{EA}{L}(\Delta p_1 - \Delta p_0) \tag{2}$$

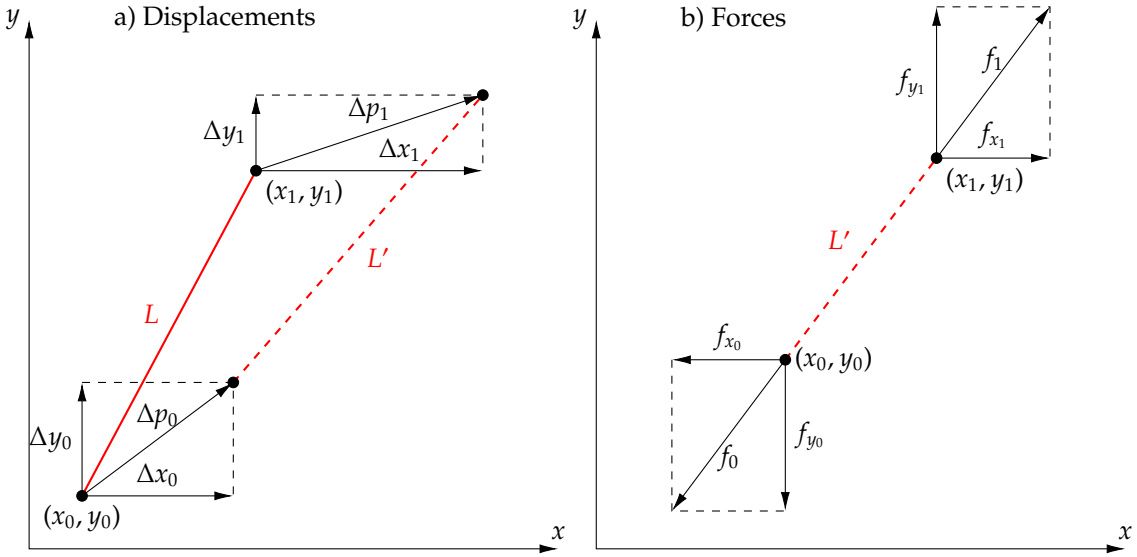

Figure 3: Displacements (a) and Stress Forces (b) in an Element

In fact the method provides only approximated solution since it assumes that angles between elements stay the same after applying external forces. In reality such approach is accepted as long as deformations are small.

Using matrix notation we isolate local stiffness matrix **k**.

$$\begin{bmatrix} f_0 \\ f_1 \end{bmatrix} = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \Delta p_0 \\ \Delta p_1 \end{bmatrix} \tag{3}$$

$$\mathbf{k} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{4}$$

The stiffness matrix represents geometry in an algebraic form. Final form of matrix equation in local coordinates shown below. Lower case bold letters reserved for local coordinates.

$$\mathbf{f} = \mathbf{k}\Delta\mathbf{p} \tag{5}$$

3.3 Coordinate Transformation

In previous chapter we described relations in local coordinates. In order to aggregate all elements into global system a number of transformations have to take place.

3.3.1 Global and Local Coordinates Relations

The figure 3a can be used in order to describe local and global geometrical relations.

$$L = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2} \tag{6}$$

$$c_x = \cos\theta_x = \frac{x_1 - x_0}{L} \tag{7}$$

$$c_y = \cos\theta_y = \frac{y_1 - y_0}{L} \tag{8}$$

$$c_z = \cos\theta_z = \frac{z_1 - z_0}{L} \tag{9}$$

3.3.2 The Point Displacement Relations

The displacement from the figure 3a for the point p_0 is broken down on the figure 4.

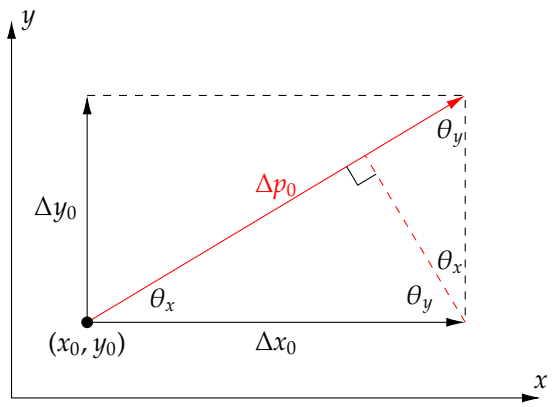

Figure 4: Displacement of the Point p_0 in Details

From congruence of triangles as on the figure 4 we obtain

$$\Delta p_0 = \Delta x_0 \cos\theta_x + \Delta y_0 \cos\theta_y + \Delta z_0 \cos\theta_z \tag{10}$$

$$\Delta p_1 = \Delta x_1 \cos\theta_x + \Delta y_1 \cos\theta_y + \Delta z_1 \cos\theta_z \tag{11}$$

In other words

$$\begin{bmatrix} \Delta p_0 \\ \Delta p_1 \end{bmatrix} = \begin{bmatrix} c_x & c_y & c_z & 0 & 0 & 0 \\ 0 & 0 & 0 & c_x & c_y & c_z \end{bmatrix} \begin{bmatrix} \Delta x_0 \\ \Delta y_0 \\ \Delta z_0 \\ \Delta x_1 \\ \Delta y_1 \\ \Delta z_1 \end{bmatrix} \tag{12}$$

or simpler

$$\Delta \mathbf{p} = \mathbf{T}\Delta \mathbf{P} \tag{13}$$

The equation 13 describes displacement relation between local and global coordinates.

3.3.3 Global and Local Forces Relations

Forces given directly from geometry on the figure 3b

$$f_{x_0} = f_0 \cos\theta_x \tag{14}$$
$$f_{y_0} = f_0 \cos\theta_y \tag{15}$$
$$f_{z_0} = f_0 \cos\theta_z \tag{16}$$
$$f_{x_1} = f_1 \cos\theta_x \tag{17}$$
$$f_{y_1} = f_1 \cos\theta_y \tag{18}$$
$$f_{z_1} = f_1 \cos\theta_z \tag{19}$$

or in a matrix form

$$\begin{bmatrix} f_{x_0} \\ f_{y_0} \\ f_{z_0} \\ f_{x_1} \\ f_{y_1} \\ f_{z_1} \end{bmatrix} = \begin{bmatrix} c_x & 0 \\ c_y & 0 \\ c_z & 0 \\ 0 & c_x \\ 0 & c_y \\ 0 & c_z \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \end{bmatrix} \tag{20}$$

$$\mathbf{F} = \mathbf{T}^\mathbf{T}\mathbf{f} \tag{21}$$

The equation 21 describes forces relation between local and global coordinates.

3.4 The Stiffness Matrix of an Element in Global Coordinates

Equations obtained in previous chapter: Hook's law, displacement and force relations between local and global coordinates are repeated below.

$$\mathbf{f} = \mathbf{k}\Delta \mathbf{p} \tag{22}$$
$$\Delta \mathbf{p} = \mathbf{T}\Delta \mathbf{P} \tag{23}$$
$$\mathbf{F} = \mathbf{T}^\mathbf{T}\mathbf{f} \tag{24}$$

It can be transformed into

$$\mathbf{f} = \mathbf{k}\Delta \mathbf{p} \tag{25}$$
$$\mathbf{f} = \mathbf{k}\mathbf{T}\Delta \mathbf{P} \tag{26}$$
$$\mathbf{T}^\mathbf{T}\mathbf{f} = \mathbf{T}^\mathbf{T}\mathbf{k}\mathbf{T}\Delta \mathbf{P} \tag{27}$$
$$\mathbf{F} = \mathbf{T}^\mathbf{T}\mathbf{k}\mathbf{T}\Delta \mathbf{P} \tag{28}$$

Finally the extracted version of global stiffness matrix $\mathbf{K}$ in global coordinates takes form

$$\mathbf{K} = \mathbf{T}^\mathbf{T}\mathbf{k}\mathbf{T} = \frac{EA}{L}\begin{bmatrix} c_x^2 & c_xc_y & c_xc_z & -c_x^2 & -c_xc_y & -c_xc_z \\ c_xc_y & c_y^2 & c_yc_z & -c_xc_y & -c_y^2 & -c_yc_z \\ c_xc_z & c_yc_z & c_z^2 & -c_xc_z & -c_yc_z & -c_z^2 \\ -c_x^2 & -c_xc_y & -c_xc_z & c_x^2 & c_xc_y & c_xc_z \\ -c_xc_y & -c_y^2 & -c_yc_z & c_xc_y & c_y^2 & c_yc_z \\ -c_xc_z & -c_yc_z & -c_z^2 & c_xc_z & c_yc_z & c_z^2 \end{bmatrix} \tag{29}$$

so the main equation of this work is formed as

$$\mathbf{F} = \mathbf{K}\Delta \mathbf{P} \tag{30}$$

and describes relation of global forces and deformation of whole truss geometry.

3.5 Global Stiffness Matrix Aggregation

Let's form equation 30 for an element $e_2(p_0, p_2)$ in Global Stiffness Matrix. The element index for cosines is omitted for simplicity (c_x instead of c_{2_x} etc.).

$$\frac{EA}{L}\begin{bmatrix} c_x^2 & c_xc_y & c_xc_z & \dots & -c_x^2 & -c_xc_y & -c_xc_z & \dots \\ c_xc_y & c_y^2 & c_yc_z & \dots & -c_xc_y & -c_y^2 & -c_yc_z & \dots \\ c_xc_z & c_yc_z & c_z^2 & \dots & -c_xc_z & -c_yc_z & -c_z^2 & \dots \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots \\ -c_x^2 & -c_xc_y & -c_xc_z & \dots & c_x^2 & c_xc_y & c_xc_z & \dots \\ -c_xc_y & -c_y^2 & -c_yc_z & \dots & c_xc_y & c_y^2 & c_yc_z & \dots \\ -c_xc_z & -c_yc_z & -c_z^2 & \dots & c_xc_z & c_yc_z & c_z^2 & \dots \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots \end{bmatrix}\begin{bmatrix} \Delta x_0 \\ \Delta y_0 \\ \Delta z_0 \\ \vdots \\ \vdots \\ \vdots \\ \Delta x_2 \\ \Delta y_2 \\ \Delta z_2 \\ \vdots \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} f_{x_0} \\ f_{y_0} \\ f_{z_0} \\ \vdots \\ \vdots \\ \vdots \\ f_{x_2} \\ f_{y_2} \\ f_{z_2} \\ \vdots \\ \vdots \\ \vdots \end{bmatrix} \tag{31}$$

By substituting parts of above equation by

$$k_2 = \frac{EA}{L}\begin{bmatrix} c_x^2 & c_xc_y & c_xc_z \\ c_xc_y & c_y^2 & c_yc_z \\ c_xc_z & c_yc_z & c_z^2 \end{bmatrix} \qquad \Delta p_i = \begin{bmatrix} \Delta x_i \\ \Delta y_i \\ \Delta z_i \end{bmatrix} \qquad f_i = \begin{bmatrix} f_{x_i} \\ f_{y_i} \\ f_{z_i} \end{bmatrix} \tag{32}$$

we obtain simpler form ready for further analysis

$$\begin{bmatrix} k_2 & \dots & -k_2 & \dots \\ \vdots & \dots & \vdots & \dots \\ -k_2 & \dots & k_2 & \dots \\ \vdots & \dots & \vdots & \dots \end{bmatrix}\begin{bmatrix} \Delta p_0 \\ \vdots \\ \Delta p_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} f_0 \\ \vdots \\ f_2 \\ \vdots \end{bmatrix} \tag{33}$$

$$\mathbf{K_2}\Delta\mathbf{P} = \mathbf{F_2} \tag{34}$$

The sum of all directional forces based on the condition of equilibrium in a point is equal zero

$$\sum \mathbf{F_x} = 0 \qquad \sum \mathbf{F_y} = 0 \qquad \sum \mathbf{F_z} = 0 \tag{35}$$

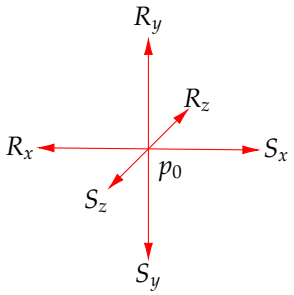

Figure 5: Force Equilibrium in the Point p_0, **R**eactions and **S**tresses

and displacement in the point is the same for all elements based on it. So we can write down

$$\sum_{e=1}^{n} \mathbf{K_e}\Delta\mathbf{P} = \sum_{e=1}^{n} \mathbf{F_e} \qquad \text{e - index of an element} \tag{36}$$

Finally the simplified form of **K** for all elements (aggregation with collocation)

$$\begin{bmatrix} k_0 + k_2 + k_3 & -k_0 & -k_2 & -k_3 \\ -k_0 & k_0 + k_1 + k_4 & -k_1 & -k_4 \\ -k_2 & -k_1 & k_1 + k_2 + k_5 & -k_5 \\ -k_3 & -k_4 & -k_5 & k_3 + k_4 + k_5 \end{bmatrix}\begin{bmatrix} \Delta p_0 \\ \Delta p_1 \\ \Delta p_2 \\ \Delta p_3 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} \tag{37}$$

The aggregated version of equation 31 is omitted here due to a poor readability.

The numerical version of the stiffness matrix with no forces applied as generated by the concrete example

$$\begin{bmatrix}
222.2 & . & . & . & . & . & -222.2 & . & . & . & . & . \\
. & 250 & . & . & -250 & . & . & . & . & . & . & . \\
. & . & 166.7 & . & . & . & . & . & . & . & . & -166.7 \\
. & . & . & 92.78 & -82.47 & . & -92.78 & 82.47 & . & . & . & . \\
. & -250 & . & -82.47 & 366 & -64 & 82.47 & -73.31 & . & . & -42.67 & 64 \\
. & . & . & . & -64 & 96.01 & . & . & . & . & 64 & -96.01 \\
-222.2 & . & . & -92.78 & 82.47 & . & 363 & -82.47 & -64 & -48 & . & 64 \\
. & . & . & 82.47 & -73.31 & . & -82.47 & 73.31 & . & . & . & . \\
. & . & . & . & . & . & -64 & . & 85.33 & 64 & . & -85.33 \\
. & . & . & . & . & . & -48 & . & 64 & 48 & . & -64 \\
. & . & . & . & -42.67 & 64 & . & . & . & . & 42.67 & -64 \\
. & . & -166.7 & . & 64 & -96.01 & 64 & . & -85.33 & -64 & -64 & 348
\end{bmatrix}
\begin{bmatrix} . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \end{bmatrix}
=
\begin{bmatrix} . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \end{bmatrix} \tag{38}$$

3.6 Fixing a Point Displacement

Determinant of matrix **K** constructed in previous chapter is equal zero.

$$det(\mathbf{K}) = 0 \tag{39}$$

It means there is no unique solution and the truss is statically unstable. Applying external force does not cause any reaction. In order to stabilise it some coordinates in some points of geometry have to be locked. It can be understood as fixing the truss to the wall. The procedure is as follows:

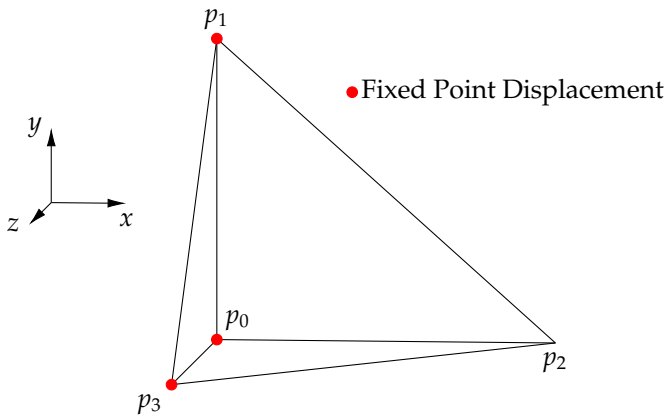

Figure 6: Fixing Tetrahedron to the Wall

Consider linear equations with exactly one solution

$$a_1x_0 + b_1x_1 + c_1x_2 = d_1 \tag{40}$$

$$a_2x_0 + b_2x_1 + c_2x_2 = d_2 \tag{41}$$

$$a_3x_0 + b_3x_1 + c_3x_2 = d_3 \tag{42}$$

or

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} \tag{43}$$

Fixing a point displacement requires setting selected variable to zero (no point displacement in this direction). Let's choose $x_1 = 0$. In order to preserve equality the constant d_2 must be freed. Disregarding zeros and moving new variable to the left side we obtain:

$$a_1 x_0 + c_1 x_2 = d_1 \tag{44}$$

$$a_2 x_0 + c_2 x_2 - d_2 = 0 \tag{45}$$

$$a_3 x_0 + c_3 x_2 = d_3 \tag{46}$$

$$\begin{bmatrix} a_1 & 0 & c_1 \\ a_2 & -1 & c_2 \\ a_3 & 0 & c_3 \end{bmatrix} \begin{bmatrix} x_0 \\ d_2 \\ x_2 \end{bmatrix} = \begin{bmatrix} d_1 \\ 0 \\ d_3 \end{bmatrix} \tag{47}$$

Concluding - in order to fix some degrees of freedom the corresponding column of the stiffness matrix has to be zeroed and a fixed variable replaced with -1. Corresponding cell in solution vector will hold *freed* constant.

$$\begin{bmatrix} -1 & . & . & . & . & . & -222.2 & . & . & . & . & . \\ . & 250 & . & . & -250 & . & . & . & . & . & . & . \\ . & . & -1 & . & . & . & . & . & . & . & . & . \\ . & . & . & -1 & -82.47 & . & -92.78 & 82.47 & . & . & . & . \\ . & -250 & . & . & 366 & . & 82.47 & -73.31 & . & . & . & . \\ . & . & . & . & -64 & -1 & . & . & . & . & . & . \\ . & . & . & . & 82.47 & . & 363 & -82.47 & -64 & . & . & . \\ . & . & . & . & -73.31 & . & -82.47 & 73.31 & . & . & . & . \\ . & . & . & . & . & . & -64 & . & 85.33 & . & . & . \\ . & . & . & . & . & . & -48 & . & 64 & -1 & . & . \\ . & . & . & . & -42.67 & . & . & . & . & . & -1 & . \\ . & . & . & . & 64 & . & 64 & . & -85.33 & . & . & -1 \end{bmatrix} \begin{bmatrix} . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \end{bmatrix} = \begin{bmatrix} . \\ . \\ . \\ . \\ . \\ . \\ . \\ 30 \\ 30 \\ . \\ . \\ . \end{bmatrix} \tag{48}$$

3.7 Implementation

```
const int dof_no = 3 * point_no; // degrees of freedom
double K[dof_no][dof_no] = { { 0.0 } }; // stiffness matrix
double F[dof_no] = { 0.0 }; // force vector

void populate_equation()
{
  // init F, K
  for(int i = 0; i < point_no; i++)
  {
    const Point3D& fix = fix_list[i];
    Point3D& force = force_list[i];
    int px = 3 * i + 0;
    int py = 3 * i + 1;
    int pz = 3 * i + 2;

    // reactions in K (in place of fixed displacement)
    if(fix.x) K[px][px] = -1;
    else F[px] = force.x;

    if(fix.y) K[py][py] = -1;
    else F[py] = force.y;

    if(fix.z) K[pz][pz] = -1;
    else F[pz] = force.z;
  }

  // compose K - stiffness matrix
  for(int i = 0; i < element_no; i++)
  {
    const double EA = 1000; // Young * Area
```

```
const Element& element = element_list[i];
const Point3D& point1 = point_list[element.p1];
const Point3D& point2 = point_list[element.p2];
const Point3D& fix1 = fix_list[element.p1];
const Point3D& fix2 = fix_list[element.p2];
int p1x = 3 * element.p1 + 0;
int p1y = 3 * element.p1 + 1;
int p1z = 3 * element.p1 + 2;
int p2x = 3 * element.p2 + 0;
int p2y = 3 * element.p2 + 1;
int p2z = 3 * element.p2 + 2;
double dx = point2.x - point1.x;
double dy = point2.y - point1.y;
double dz = point2.z - point1.z;
double l = ::sqrt(dx * dx + dy * dy + dz * dz);
double cx = dx / l;
double cy = dy / l;
double cz = dz / l;
double cxxEAl = cx * cx * EA / l;
double cyyEAl = cy * cy * EA / l;
double czzEAl = cz * cz * EA / l;
double cxyEAl = cx * cy * EA / l;
double cxzEAl = cx * cz * EA / l;
double cyzEAl = cy * cz * EA / l;

if(!fix1.x)
{
  K[p1x][p1x] += cxxEAl;
  K[p1y][p1x] += cxyEAl;
  K[p1z][p1x] += cxzEAl;
  K[p2x][p1x] -= cxxEAl;
  K[p2y][p1x] -= cxyEAl;
  K[p2z][p1x] -= cxzEAl;
}
if(!fix1.y)
{
  K[p1x][p1y] += cxyEAl;
  K[p1y][p1y] += cyyEAl;
  K[p1z][p1y] += cyzEAl;
  K[p2x][p1y] -= cxyEAl;
  K[p2y][p1y] -= cyyEAl;
  K[p2z][p1y] -= cyzEAl;
}
if(!fix1.z)
{
  K[p1x][p1z] += cxzEAl;
  K[p1y][p1z] += cyzEAl;
  K[p1z][p1z] += czzEAl;
  K[p2x][p1z] -= cxzEAl;
  K[p2y][p1z] -= cyzEAl;
  K[p2z][p1z] -= czzEAl;
}
if(!fix2.x)
{
  K[p1x][p2x] -= cxxEAl;
  K[p1y][p2x] -= cxyEAl;
  K[p1z][p2x] -= cxzEAl;
  K[p2x][p2x] += cxxEAl;
  K[p2y][p2x] += cxyEAl;
  K[p2z][p2x] += cxzEAl;
}
if(!fix2.y)
{
```

```
        K[p1x][p2y] -= cxyEAl;
        K[p1y][p2y] -= cyyEAl;
        K[p1z][p2y] -= cyzEAl;
        K[p2x][p2y] += cxyEAl;
        K[p2y][p2y] += cyyEAl;
        K[p2z][p2y] += cyzEAl;
      }
      if(!fix2.z)
      {
        K[p1x][p2z] -= cxzEAl;
        K[p1y][p2z] -= cyzEAl;
        K[p1z][p2z] -= czzEAl;
        K[p2x][p2z] += cxzEAl;
        K[p2y][p2z] += cyzEAl;
        K[p2z][p2z] += czzEAl;
      }
    }
  }
```

4 Solving Linear Equations by LU Decomposition Method

4.1 Explanation

The method takes advantage of the fact that solving equation where matrix is triangular is straightforward. All cells above (or below) diagonal are equal zero and solution can be achieved in one iteration. The method at first breaks down the matrix into a multiplication of 2 triangular ones and then calculates sub-equations as follows.

Consider linear equations with exactly one solution

$$a_1x_0 + b_1x_1 + c_1x_2 = d_1 \tag{49}$$

$$a_2x_0 + b_2x_1 + c_2x_2 = d_2 \tag{50}$$

$$a_3x_0 + b_3x_1 + c_3x_2 = d_3 \tag{51}$$

or

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} \tag{52}$$

$$\mathbf{K}\Delta\mathbf{P} = \mathbf{F} \tag{53}$$

LU Decomposition Method is one of the simplest, though memory hungry solver. Let's decompose matrix **A** into 2 triangular matrices LU (lower and upper).

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \tag{54}$$

The equation can be rewritten as

$$\mathbf{LU}\Delta\mathbf{P} = \mathbf{F} \tag{55}$$

When L and U are calculated the auxiliary vector Z can be simply solved

$$\mathbf{LZ} = \mathbf{F} \tag{56}$$

Finally the vector $\Delta\mathbf{P}$ can be solved in similar way

$$\mathbf{U}\Delta\mathbf{P} = \mathbf{Z} \tag{57}$$

4.2 Solving an Example

Let's solve a simple example.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 14 & 19 \\ 5 & 58 & 80 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 10 \\ 59 \\ 211 \end{bmatrix} \tag{58}$$

LU Decomposition

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 14 & 19 \\ 5 & 58 & 80 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \tag{59}$$

It can be noticed that general direction of solving cells of matrices L and U is as depicted on figure 7.

$$\begin{array}{lcll} 1 = 1 * u_{11} & \Rightarrow & & \Rightarrow \quad u_{11} = 1 \\ 2 = 1 * u_{12} & \Rightarrow & & \Rightarrow \quad u_{12} = 2 \\ 3 = 1 * u_{13} & \Rightarrow & & \Rightarrow \quad u_{13} = 3 \\ 4 = l_{21} u_{11} & \Rightarrow & 4 = l_{21} * 1 & \Rightarrow \quad l_{21} = 4 \\ 5 = l_{31} u_{11} & \Rightarrow & 5 = l_{31} * 1 & \Rightarrow \quad l_{31} = 5 \\ 14 = l_{21} u_{12} + u_{22} & \Rightarrow & 14 = 4 * 2 + u_{22} & \Rightarrow \quad u_{22} = 6 \\ 19 = l_{21} u_{13} + u_{23} & \Rightarrow & 19 = 4 * 3 + u_{23} & \Rightarrow \quad u_{23} = 7 \\ 58 = l_{31} u_{12} + l_{32} u_{22} & \Rightarrow & 58 = 5 * 2 + l_{32} * 6 & \Rightarrow \quad l_{32} = 8 \\ 80 = l_{31} u_{13} + l_{32} u_{23} + u_{33} & \Rightarrow & 80 = 5 * 3 + 8 * 7 + u_{33} & \Rightarrow \quad u_{33} = 9 \end{array} \tag{60}$$

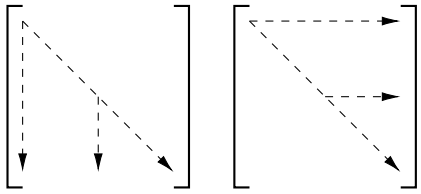

Figure 7: Direction of the Decomposition in Matrices L and U

Verification of LU Decomposition

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 14 & 19 \\ 5 & 58 & 80 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 5 & 8 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 6 & 7 \\ 0 & 0 & 9 \end{bmatrix} \tag{61}$$

Solving the auxiliary Z vector

$$\begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 5 & 8 & 1 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 10 \\ 59 \\ 211 \end{bmatrix} \tag{62}$$

$$\begin{array}{lcll} 1 * z_0 = 10 & \Rightarrow & & \Rightarrow \quad z_0 = 10 \\ 4 * z_0 + 1 * z_1 = 59 & \Rightarrow & 4 * 10 + z_1 = 59 & \Rightarrow \quad z_1 = 19 \\ 5 * z_0 + 8 * z_1 + 1 * z_2 = 211 & \Rightarrow & 5 * 10 + 8 * 19 + 1 * z_2 = 211 & \Rightarrow \quad z_2 = 9 \end{array} \tag{63}$$

Verification

$$\begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 5 & 8 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 19 \\ 9 \end{bmatrix} = \begin{bmatrix} 10 \\ 59 \\ 211 \end{bmatrix} \tag{64}$$

The final solution

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 6 & 7 \\ 0 & 0 & 9 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 10 \\ 19 \\ 9 \end{bmatrix} \tag{65}$$

$$
\begin{array}{lcll}
9 * x_2 = 9 & \Rightarrow & & \Rightarrow \quad x_2 = 1 \\
6 * x_1 + 7 * x_2 = 19 & \Rightarrow & 6 * x_1 + 7 * 1 = 19 & \Rightarrow \quad x_1 = 2 \\
1 * x_0 + 2 * x_1 + 3 * x_2 = 10 & \Rightarrow & 1 * x_0 + 2 * 2 + 3 * 1 = 10 & \Rightarrow \quad x_0 = 3
\end{array} \tag{66}
$$

Verification

$$
\begin{bmatrix} 1 & 2 & 3 \\ 0 & 6 & 7 \\ 0 & 0 & 9 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 19 \\ 9 \end{bmatrix} \tag{67}
$$

Final verification of the original equation

$$
\begin{bmatrix} 1 & 2 & 3 \\ 4 & 14 & 19 \\ 5 & 58 & 80 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 59 \\ 211 \end{bmatrix} \tag{68}
$$

4.3 Pitfalls of Direct Methods

The present example uses LU Decomposition for solving matrix equations. The main downside of such approach is that data blocks are composed mostly with zeros. It is very inefficient management of memory and can be used only for simple geometries. Bigger examples may not fit into memory of modern computers. More realistic model should be solved by one of sparse matrix methods. The most promising of them is Conjugate Gradient Method. Further work on next version of this document will be focused on it.

4.4 The Concrete Example Result Presentation

The below matrix equation presents the stiffness matrix, the displacement vector and external forces applied in selected points. Some cells where -1 is shown are used for fixed displacements and were chosen to compute reaction forces in corresponding displacement vector. The simplicity dictates such approach for price of loosing symmetry of matrix.

The data in the form

$$
\mathbf{K}\Delta\mathbf{P} = \mathbf{F} \tag{69}
$$

is generated below as a result of computation of the main example

$$
\begin{bmatrix}
-1 & . & . & . & . & . & -222.2 & . & . & . & . & . \\
. & 250 & . & . & -250 & . & . & . & . & . & . & . \\
. & . & -1 & . & . & . & . & . & . & . & . & . \\
. & . & . & -1 & -82.47 & . & -92.78 & 82.47 & . & . & . & . \\
. & -250 & . & . & 366 & . & 82.47 & -73.31 & . & . & . & . \\
. & . & . & . & -64 & -1 & . & . & . & . & . & . \\
. & . & . & . & 82.47 & . & 363 & -82.47 & -64 & . & . & . \\
. & . & . & . & -73.31 & . & -82.47 & 73.31 & . & . & . & . \\
. & . & . & . & . & . & -64 & . & 85.33 & . & . & . \\
. & . & . & . & . & . & -48 & . & 64 & -1 & . & . \\
. & . & . & . & -42.67 & . & . & . & . & . & -1 & . \\
. & . & . & . & 64 & . & 64 & . & -85.33 & . & . & -1
\end{bmatrix}
\begin{bmatrix} -56.25 \\ 0.7031 \\ . \\ 33.75 \\ 0.7031 \\ -45 \\ 0.2531 \\ 1.397 \\ 0.5414 \\ 22.5 \\ -30 \\ 15 \end{bmatrix}
=
\begin{bmatrix} . \\ . \\ . \\ . \\ . \\ . \\ . \\ 30 \\ 30 \\ . \\ . \\ . \end{bmatrix} \tag{70}
$$

4.5 Implementation of LU Decomposition Method

```
double dP[dof_no] = { 0.0 }; // K * dP = F, delta P - solution vector

// solving K * dP = F by LU method
void calculate_equation()
{
  double L[dof_no][dof_no] = { { 0.0 } }; // lower matrix
```

```
double U[dof_no][dof_no] = { { 0.0 } }; // upper matrix
double Z[dof_no] = { 0.0 }; // auxiliary vector

// init L := I
for(int i = 0; i < dof_no; i++)
{
  L[i][i] = 1.0;
}

// find L, U where L * U = K
for(int i1 = 0; i1 < dof_no; i1++)
{
  double acc = 0.0;
  for(int i2 = 0; i2 < dof_no; i2++)
    acc += L[i1][i2] * U[i2][i1];
  U[i1][i1] = K[i1][i1] - acc;

  for(int i2 = i1 + 1; i2 < dof_no; i2++)
  {
    acc = 0.0;
    for(int i3 = 0; i3 < i1; i3++)
      acc += L[i1][i3] * U[i3][i2];
    U[i1][i2] = K[i1][i2] - acc;

    acc = 0.0;
    for(int i3 = 0; i3 < i1; i3++)
      acc += L[i2][i3] * U[i3][i1];
    L[i2][i1] = (K[i2][i1] - acc) / U[i1][i1];
  }
}

// finally find result
for(int i1 = 0; i1 < dof_no; i1++)
{
  // find Z where L * Z = F
  double acc = 0.0;
  for(int i2 = 0; i2 < i1; i2++)
    acc += L[i1][i2] * Z[i2];
  Z[i1] = F[i1] - acc;
}
for(int i1 = dof_no - 1; i1 >= 0; i1--)
{
  // find dP where U * dP = Z
  double acc = 0.0;
  for(int i2 = i1; i2 < dof_no; i2++)
    acc += U[i1][i2] * dP[i2];
  dP[i1] = (Z[i1] - acc) / U[i1][i1];
}

// copy to global output
for(int i = 0; i < point_no; i++)
{
  const Point3D& point = point_list[i];
  const Point3D& fix = fix_list[i];
  Point3D& force = force_list[i];
  Point3D& output = output_list[i];

  int px = 3 * i + 0;
  int py = 3 * i + 1;
  int pz = 3 * i + 2;
  output = point;

  if(fix.x) force.x = dP[px];
```

```
      else output.x += dP[px];

      if(fix.y) force.y = dP[py];
      else output.y += dP[py];

      if(fix.z) force.z = dP[pz];
      else output.z += dP[pz];
    }
  }
```

5 The 3D Data Presentation in the Documentation

This chapter tries to explain how to present three-dimensional data in the document. Using only x, y and removing z coordinate is not a good solution since we loose part of the information and presented figures are not clear. Rotation is much better as all of original coordinates can be used and after transformation the new coordinate z' can be discarded without loosing general understanding.

5.1 Point Rotation

Let's consider rotation of a point in coordinates x, y

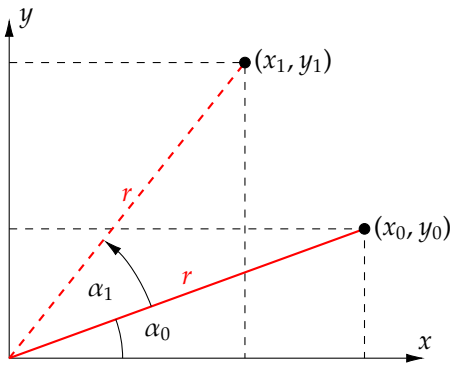

Figure 8: Point Rotation

From geometry on the figure 8 we can obtain

$$x_0 = r\cos\alpha_0 \tag{71}$$

$$y_0 = r\sin\alpha_0 \tag{72}$$

$$x_1 = r\cos(\alpha_0+\alpha_1) = r\cos\alpha_0\cos\alpha_1 - r\sin\alpha_0\sin\alpha_1 \tag{73}$$

$$y_1 = r\sin(\alpha_0+\alpha_1) = r\sin\alpha_0\cos\alpha_1 + r\cos\alpha_0\sin\alpha_1 \tag{74}$$

hence

$$x_1 = x_0\cos\alpha_1 - y_0\sin\alpha_1 \tag{75}$$

$$y_1 = x_0\sin\alpha_1 + y_0\cos\alpha_1 \tag{76}$$

Changes on the axis z can be expressed simply by $z_1 = z_0$. Constructing three-dimensional matrix we obtain rotation operator around the axis z

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} \cos\alpha_1 & -\sin\alpha_1 & 0 \\ \sin\alpha_1 & \cos\alpha_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{77}$$

$$\mathbf{P_1} = \mathbf{R_z}\mathbf{P_0} \tag{78}$$

5.2 An Example of Rotated Tetrahedron

The example rotates all points of geometry around the axis y and then x. It can be expressed as

$$\mathbf{P_1} = \mathbf{R_x}\mathbf{R_y}\mathbf{P_0} \tag{79}$$

In details

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_x & -s_x \\ 0 & s_x & c_x \end{bmatrix} \begin{bmatrix} c_y & 0 & -s_y \\ 0 & 1 & 0 \\ s_y & 0 & c_y \end{bmatrix} \tag{80}$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} c_y & 0 & -s_y \\ -s_x s_y & c_x & -s_x c_y \\ c_x s_y & s_x & c_x c_y \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{81}$$

Finally the coordinate z_1 is cut off as it does not have representation in 2D. Visualized results are presented on the figure 9.

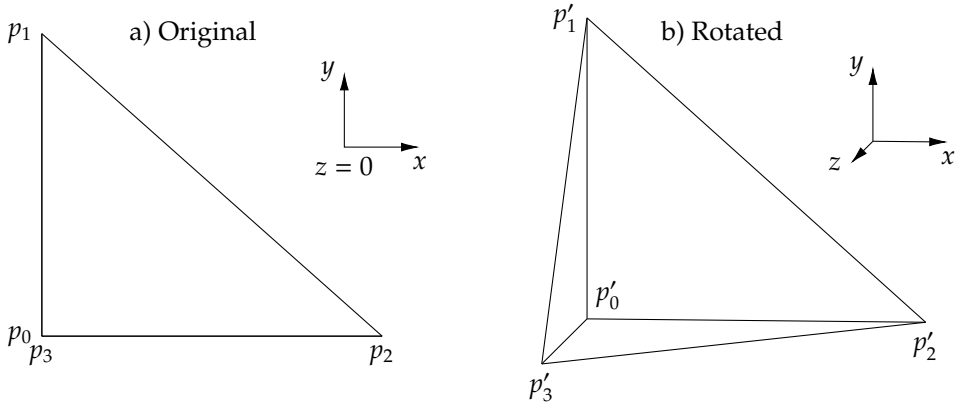

Figure 9: Original and Rotated Tetrahedron Example

5.3 Implementation of the Rotation Procedure

```
const double rot_x = .1; // rotation angle around axis X
const double rot_y = .1; // rotation angle around axis Z
const double c_x = ::cos(rot_x);
const double s_x = ::sin(rot_x);
const double c_y = ::cos(rot_y);
const double s_y = ::sin(rot_y);

// transform/rotate point from 3D => 2D
Point2D rotate(const Point3D& p)
{
  return Point2D
  (
    c_y * p.x - s_y * p.z,
    - s_x * s_y * p.x + c_x * p.y - s_x * c_y * p.z
  );
}
```

6 Summary

Finally the article described all practical aspects of the finite method presentation. Starting with geometry definition, theoretical method explanation, solving matrix equations and 3D data presentation. Reader could confront mathematical aspects with practical implementation for better understanding.