INTRODUCTION

This assignment deals with the analysis of 2-Dimensional truss elements using Finite Element Methods. A truss element is a bar which can resist only axial forces (compressive or tensile) and can deform only in the axial direction. It will not be able to carry transverse loads or bending moments. These truss elements are joined by pins and can rotate about them.

In planar truss analysis, each of the two nodes can have components of displacement parallel to the X and Y axes. In three dimensional truss analysis, each of the nodes can have three displacement components along the X, Y and Z axes. In all these cases we assume the members to be linearly elastic and hence Hooke's law can be applied.

THEORY

Consider a pin-jointed bar element where the local X-Axis is taken in the axial direction of the element. Origin is located at local node 1 and linear displacement is assumed.

$$u(x) = u_1^1 + (u_1^2 - u_1^1)\frac{x}{l}$$

$$[u(x)] = [N]\{u\}$$

where,

$$[N] = \left[(1 - \frac{x}{l}) \quad \frac{x}{l} \right]$$

$$\{u\} = \begin{Bmatrix} u_1^1 \\ u_1^2 \end{Bmatrix}$$

u_1^1 and u_1^2 represent the displacements in the local coordinate system. The axial strain can be expressed as

$$[\varepsilon_{xx}] = [B]\{u\}$$
$$[B] = \begin{bmatrix} \frac{-1}{l} & \frac{1}{l} \end{bmatrix}$$

The stress strain relationship is given by

$$\{\sigma_{xx}\} = [D]\{\varepsilon_{xx}\}$$

where [*D*] = [*E*] and E is the young's modulus of the material. The stiffness matrix of the element in the local coordinates is got by

$$[K]^e = \iiint_{V^e} [B]^T [D][B] dV$$
$$= A \int_{x=0}^{l} \begin{Bmatrix} \frac{-1}{l} \\ \frac{1}{l} \end{Bmatrix} E \begin{Bmatrix} \frac{-1}{l} & \frac{1}{l} \end{Bmatrix} dx$$
$$= \frac{AE}{l} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

where A is the area of cross section of the bar.

To find the Global Stiffness matrix, the local stiffness matrix has to be multiplied by a transformation matrix. In general, the element under consideration will be one of the elements of the truss. Let the local nodes correspond to nodes i and in the global truss. The local displacements can be resolved into global components parallel to the global X, Y and Z axes. The two displacements are related as

$$\{u\}^e = [\lambda]\{u\}^g$$
$$[\lambda] = \begin{bmatrix} l_{ij} & m_{ij} & n_{ij} & 0 & 0 & 0 \\ 0 & 0 & 0 & l_{ij} & m_{ij} & n_{ij} \end{bmatrix}$$
$$\{u\}^g = \begin{Bmatrix} u_1^i \\ u_2^i \\ u_3^i \\ u_1^j \\ u_2^j \\ u_3^j \end{Bmatrix}$$

where $l_{ij,}$ $m_{ij,}$ n_{ij} denote the direction cosines of the angles between the line ij and the respective directions OX, OY and OZ.

The direction cosines can be calculated as

$$l_{ij} = \frac{u_1^j - u_1^2}{l}$$

$$m_{ij} = \frac{u_2^j - u_2^2}{l}$$

$$n_{ij} = \frac{u_3^j - u_3^2}{l}$$

where u's represent the coordinates of the nodes i and j . Thus the global stiffness matrix can be obtained as

$$[K]^g = [\lambda]^T [K]^e [\lambda]$$

$$= \frac{AE}{l} \begin{bmatrix} l_{ij}^2 & l_{ij}m_{ij} & l_{ij}n_{ij} & -l_{ij}^2 & -l_{ij}m_{ij} & -l_{ij}n_{ij} \\ l_{ij}m_{ij} & m_{ij}^2 & m_{ij}n_{ij} & -l_{ij}m_{ij} & -m_{ij}^2 & -m_{ij}n_{ij} \\ l_{ij}n_{ij} & m_{ij}n_{ij} & n_{ij}^2 & -l_{ij}n_{ij} & -m_{ij}n_{ij} & -n_{ij}^2 \\ -l_{ij}^2 & -l_{ij}m_{ij} & -l_{ij}n_{ij} & l_{ij}^2 & l_{ij}m_{ij} & l_{ij}n_{ij} \\ -l_{ij}m_{ij} & -m_{ij}^2 & -m_{ij}n_{ij} & l_{ij}m_{ij} & m_{ij}^2 & m_{ij}n_{ij} \\ -l_{ij}n_{ij} & -m_{ij}n_{ij} & -n_{ij}^2 & l_{ij}n_{ij} & m_{ij}n_{ij} & n_{ij}^2 \end{bmatrix}$$

After finding the displacement solution of the system, the stress induced in an element can be found as

$$\sigma_{xx} = E[B][\lambda]\{u\}$$

where all the matrices are as defined before.

For a two dimensional truss the Global stiffness matrix values get changed and it reduces to a 4×4 matrix.

$$[K]^g = \frac{AE}{l} \begin{bmatrix} l_{ij}^2 & l_{ij}m_{ij} & -l_{ij}^2 & -l_{ij}m_{ij} \\ l_{ij}m_{ij} & m_{ij}^2 & -l_{ij}m_{ij} & -m_{ij}^2 \\ -l_{ij}^2 & -l_{ij}m_{ij} & l_{ij}^2 & l_{ij}m_{ij} \\ -l_{ij}m_{ij} & -m_{ij}^2 & l_{ij}m_{ij} & m_{ij}^2 \end{bmatrix}$$

Similarly all the components referring to the third coordinate in other vectors are removed from the formulation.

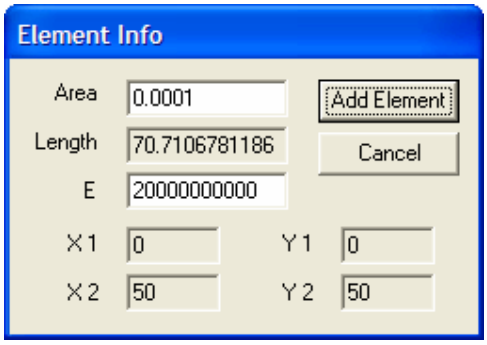
CODING

To solve the truss problem a generalized program was made using Visual C++. The outline of the steps involved in solving a problem using the program is

1. Adding Truss Elements
2. Assembling of Global Stiffness Matrix
3. Setting Boundary conditions
4. Solving the Problem
5. Output

ADDING TRUSS ELEMENTS

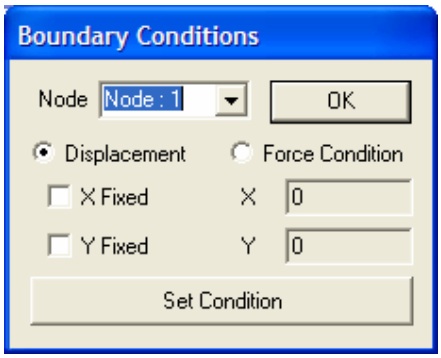
To aid in automation a graphical user interface was provided and is used to interactively create the truss. The node numbering is automatically done and a property box is used to add the values of young's modulus and area of the truss member.



The element is added into a array called ElemConnect and nodes to an array called NodeCood.

BOUNDARY CONDITIONS

The boundary conditions are also set interactively by using the boundary conditions dialog box. The user can choose between the displacement and force condition. If Displacement is chosen by the user then the corresponding element in the



diagonal of the global stiffness matrix is made a very high number (10^{100}). This number is multiplied by the node number to vary the big number. This was done because the Global stiffness matrix was singular sometimes if the same big number is used.

If force condition is chosen, the corresponding value is added in the force vector. All the other forces are taken as Zero.

SOLUTION AND OUTPUT

After setting all the boundary conditions, the problem is solved and the output can be viewed as X and Y Displacements on the Output Popup window

CODE

```
class CTrussElement
{
public :
	int node1;
	int node2;
	C2DPoint node1Cood;
	C2DPoint node2Cood;
	CMatrix elemStiff;
	double area;
	double E;

//constructor
public :
	CTrussElement()
	{
		this->node1=0;
		this->node2=0;
		this->elemStiff=CMatrix(4,4);
		this->area=1;
		this->E=1;
		this->node1Cood.x=0;
		this->node1Cood.y=0;
		this->node2Cood.x=0;
		this->node2Cood.y=0;
	}
	//destructor
	~CTrussElement()
	{
	}

	//operations
	void copyTruss(CTrussElement);
};

typedef struct C2DPoint_tag
{
	double x;
	double y;
	bool fixedX;
	bool fixedY;
}C2DPoint;
int CTrussDoc::CheckDuplicateNode(C2DPoint node)
{
	int pos=-1;
	for (int i=0;i<=this->NodeCount;i++)
		if (node.x==(this->NodeCood+i)->x && node.y==(this->NodeCood+i)->y)
		{
			pos=i;
			break;
		}
	return pos;
}

C2DPoint CTrussDoc::ConvertNode(CPoint node)
{
	C2DPoint converted;
	converted.x=(double)((node.x-this->Origin.x)*this->GridScale);
	converted.y=(double)(-(node.y-this->Origin.y)*this->GridScale);
```

```
	return converted;
}

CPoint CTrussDoc::ScreenNode(C2DPoint node)
{
	CPoint converted;
	converted.x=(int)((node.x/this->GridScale+this->Origin.x));
	converted.y=(int)((-node.y/this->GridScale+this->Origin.y));
	return converted;
}

void CTrussDoc::AddNode(C2DPoint node)
{
	this->NodeCount++;
	(this->NodeCood+this->NodeCount)->x=node.x;
	(this->NodeCood+this->NodeCount)->y=node.y;
	(this->NodeCood+this->NodeCount)->fixedX=false;
	(this->NodeCood+this->NodeCount)->fixedY=false;

}

int CTrussDoc::CheckDuplicateElement(int node1,int node2)
{
	int pos=-1;
	for (int i=0;i<=this->ElemCount;i++)
		if ((node1==(this->ElemConnection+i)->node1 && node2==(this->
			ElemConnection+i )-> node2)||(node1==(this->ElemConnection+i)->node2
			&& node2==(this-> ElemConnection+i)->node1))
		{
			pos=i;
			break;
		}
	return pos;
}

void CTrussDoc::AddElement(int node1,int node2)
{
	CTrussElement truss;
	truss.node1			=node1;
	truss.node2			=node2;
	truss.node1Cood.x	=(this->NodeCood+node1)->x;
	truss.node1Cood.y	=(this->NodeCood+node1)->y;
	truss.node2Cood.x	=(this->NodeCood+node2)->x;
	truss.node2Cood.y	=(this->NodeCood+node2)->y;

	 double length=sqrt((truss.node2Cood.x-truss.node1Cood.x)*(truss.node2Cood.x-
				truss.node1Cood.x)+(truss.node2Cood.y-
				truss.node1Cood.y)*(truss.node2Cood.y-truss.node1Cood.y));

	CElementInfo Info;
	Info.m_area=.0001;
	Info.m_length=length;
	Info.m_e=2.0e10;
	Info.m_x1	=(this->NodeCood+node1)->x;
	Info.m_y1	=(this->NodeCood+node1)->y;
	Info.m_x2	=(this->NodeCood+node2)->x;
	Info.m_y2	=(this->NodeCood+node2)->y;
	if (Info.DoModal()==IDOK)
	{
		truss.E=Info.m_e;
```

```
                truss.area=Info.m_area;

                double l=(truss.node2Cood.x-truss.node1Cood.x)/length;
                double m=(truss.node2Cood.y-truss.node1Cood.y)/length;
                double c=truss.area*truss.E/length;

                truss.elemStiff.SetElement(0,0,c*l*l);
                truss.elemStiff.SetElement(1,0,c*l*m);
                truss.elemStiff.SetElement(2,0,-c*l*l);
                truss.elemStiff.SetElement(3,0,-c*l*m);

                truss.elemStiff.SetElement(0,1,c*l*m);
                truss.elemStiff.SetElement(1,1,c*m*m);
                truss.elemStiff.SetElement(2,1,-c*l*m);
                truss.elemStiff.SetElement(3,1,-c*m*m);

                truss.elemStiff.SetElement(0,2,-c*l*l);
                truss.elemStiff.SetElement(1,2,-c*l*m);
                truss.elemStiff.SetElement(2,2,c*l*l);
                truss.elemStiff.SetElement(3,2,c*l*m);

                truss.elemStiff.SetElement(0,3,-c*l*m);
                truss.elemStiff.SetElement(1,3,-c*m*m);
                truss.elemStiff.SetElement(2,3,c*l*m);
                truss.elemStiff.SetElement(3,3,c*m*m);

                this->ElemCount++;
                (this->ElemConnection+this->ElemCount)->copyTruss(truss);
                this->tools.SetInfo(CString("Truss Element Added"));
        }
        else
        {
                this->tools.SetInfo(CString("Truss Element Not Added"));
        }
}

void CTrussDoc::Assemble()
{
        this->GStiffness=CMatrix(this->NodeCount*2+2,this->NodeCount*2+2);
        for (int k=0;k<=this->ElemCount;k++)
        {
                int i=(this->ElemConnection+k)->node1;
                int j=(this->ElemConnection+k)->node2;

                this->GStiffness.SetElement(2*i  ,2*i  ,this->GStiffness.GetElement(2*i  ,2*i  ) +
                                        (this->ElemConnection+k)->elemStiff.GetElement(0,0));
                this->GStiffness.SetElement(2*i+1,2*i  ,this->GStiffness.GetElement(2*i+1,2*i  ) +
                                        (this->ElemConnection+k)->elemStiff.GetElement(1,0));
                this->GStiffness.SetElement(2*j  ,2*i  ,this->GStiffness.GetElement(2*j ,2*i  ) +
                                        (this->ElemConnection+k)->elemStiff.GetElement(2,0));
                this->GStiffness.SetElement(2*j+1,2*i  ,this->GStiffness.GetElement(2*j+1,2*i  ) +
                                        (this->ElemConnection+k)->elemStiff.GetElement(3,0));

                this->GStiffness.SetElement(2*i  ,2*i+1,this->GStiffness.GetElement(2*i,
                                        2*i+1)+(this->ElemConnection+k)-
                                        >elemStiff.GetElement(0,1));
                this->GStiffness.SetElement(2*i+1,2*i+1,this-
                                        >GStiffness.GetElement(2*i+1,2*i+1)+(this-
                                        >ElemConnection+k)->elemStiff.GetElement(1,1));
```

```
                this->GStiffness.SetElement(2*j  ,2*i+1,this->GStiffness.GetElement(2*j
                                ,2*i+1)+(this->ElemConnection+k)-
                                >elemStiff.GetElement(2,1));
                this->GStiffness.SetElement(2*j+1,2*i+1,this-
                                >GStiffness.GetElement(2*j+1,2*i+1)+(this-
                                >ElemConnection+k)->elemStiff.GetElement(3,1));

                this->GStiffness.SetElement(2*i  ,2*j  ,this->GStiffness.GetElement(2*i  ,2*j
                                )+(this->ElemConnection+k)->elemStiff.GetElement(0,2));
                this->GStiffness.SetElement(2*i+1,2*j  ,this->GStiffness.GetElement(2*i+1,2*j
                                )+(this->ElemConnection+k)->elemStiff.GetElement(1,2));
                this->GStiffness.SetElement(2*j  ,2*j  ,this->GStiffness.GetElement(2*j           ,2*j
                                )+(this->ElemConnection+k)->elemStiff.GetElement(2,2));
                this->GStiffness.SetElement(2*j+1,2*j  ,this->GStiffness.GetElement(2*j+1,2*j
                                )+(this->ElemConnection+k)->elemStiff.GetElement(3,2));

                this->GStiffness.SetElement(2*i  ,2*j+1,this->GStiffness.GetElement(2*i
                                ,2*j+1)+(this->ElemConnection+k)-
                                >elemStiff.GetElement(0,3));
                this->GStiffness.SetElement(2*i+1,2*j+1,this-
                                >GStiffness.GetElement(2*i+1,2*j+1)+(this-
                                >ElemConnection+k)->elemStiff.GetElement(1,3));
                this->GStiffness.SetElement(2*j  ,2*j+1,this->GStiffness.GetElement(2*j
                                ,2*j+1)+(this->ElemConnection+k)-
                                >elemStiff.GetElement(2,3));
                this->GStiffness.SetElement(2*j+1,2*j+1,this-
                                >GStiffness.GetElement(2*j+1,2*j+1)+(this-
                                >ElemConnection+k)->elemStiff.GetElement(3,3));
                                }
}

void CTrussDoc::BoundaryCondition()
{
        this->Assemble();
        this->Force=CMatrix(1,this->NodeCount*2+2);
        this->Disp =CMatrix(1,this->NodeCount*2+2);
        CBoundcond BC;
        BC.DoModal();
}

void CBoundcond::OnButtonSet()
{
        // TODO: Add your control notification handler code here
        this->UpdateData(true);
        CTrussDoc* pDoc=(CTrussDoc*) this->GetParentFrame()->GetActiveDocument();
        this->UpdateData(true);
        int nodeNum=this->m_node.GetCurSel();
        if (this->m_disp.GetCheck()==1)
        {
                if (this->m_fixedX==1)
                {
                        pDoc->GStiffness.SetElement(nodeNum*2,
                                nodeNum*2,1.0e100*(nodeNum+1));
                        (pDoc->NodeCood+nodeNum)->fixedX=true;
                }
                if (this->m_fixedY==1)
                {
                        pDoc->GStiffness.SetElement(nodeNum*2+1,nodeNum*2+1,
                                1.0e100*(nodeNum+1));
                        (pDoc->NodeCood+nodeNum)->fixedY=true;
```

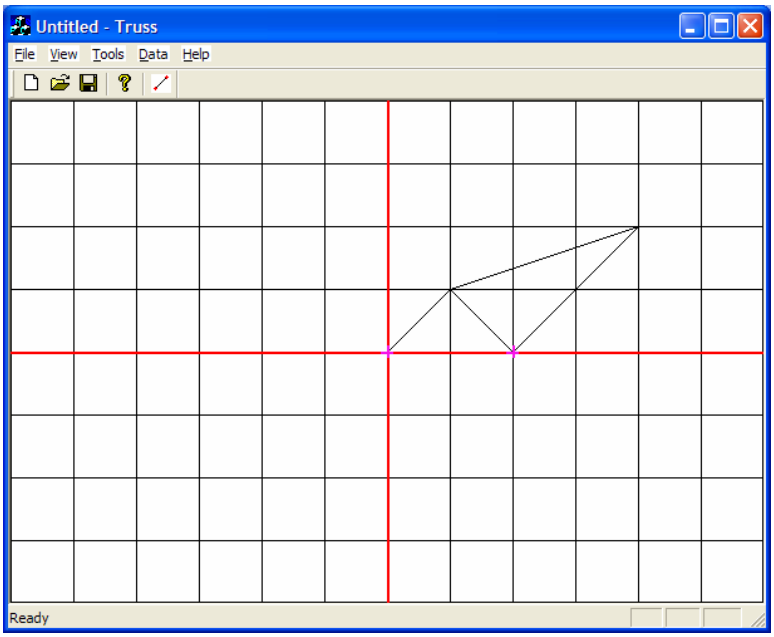
```
                }
        }
        else
        {
                pDoc->Force.SetElement(0,nodeNum*2  ,this->m_forceX);
                pDoc->Force.SetElement(0,nodeNum*2+1,this->m_forceY);
        }

}

void CTrussDoc::Solve()
{
        this->Disp=this->Force*this->GStiffness.GetInverted();
        this->solved=true;
}
```

SAMPLE PROBLEM

To check the validity of the results of the program a small sample problem is considered. The results of the problem are known and the output from the program is compared with this known output.



Node Number	X	Y
0	0.0	0.0
1	50.0	50.0
2	100.0	0.0
3	300.0	100.0

Element Number	Area	Length	Young's Modulus
0	2.0	$50\sqrt{2}$	2×10^{10}
1	2.0	$50\sqrt{2}$	2×10^{10}
2	1.0	$100\sqrt{2.5}$	2×10^{10}
3	1.0	$100\sqrt{2}$	2×10^{10}

DISPLACEMENT SOLUTION

Node Number	X	Y
0	1.00E-97	1.00E-97
1	0.02651650	0.08838835
2	-3.333333E-98	-6.666667E-98
3	0.3479025	-0.560003E

As expected the values for nodes 1 and 3 are so low that they can be neglected. These answers compare exactly with the required output.