```
[Serializable]
public class ASLVertex
{
    public ASL3d Vertex, Normal;
    public ASL2d Texture;
    public ASLVertex() { Vertex = CC.Origin; Normal = CC.K; Texture = null; }

    /// <summary>
    /// Initializes a Vertex with position only
    /// </summary>
    /// <param name="v">Position</param>
    public ASLVertex(ASL3d v) { Vertex = v.Clone(); }

    /// <summary>
    /// Initializes a Vertex with a position and a Normal
    /// </summary>
    /// <param name="v">Position</param>
    /// <param name="n">Normal</param>
    public ASLVertex(ASL3d v, ASL3d n) { Vertex = v.Clone(); Normal = n.Clone(); }

    /// <summary>
    /// Initializes a Vertex with Position, Normal and Texture coordinate
    /// </summary>
    /// <param name="v">Position</param>
    /// <param name="n">Normal</param>
    /// <param name="txt">Texture coordinates</param>
    public ASLVertex(ASL3d v, ASL3d n, ASL2d txt) { Vertex = v.Clone(); Normal = n.Clone(); Texture = txt.Clone(); }
    public ASLVertex Clone()
    {
        ASLVertex v = new ASLVertex(Vertex);
        if (Normal != null) v.Normal = Normal.Clone();
        if (Texture != null) v.Texture = Texture.Clone();
        return v;
    }
}

[Serializable]
public class ASLFace
{
    [XmlAttribute] public int v0, v1, v2, v3;
    public ASLFace() { v0 = v1 = v2 = v3 = 0; }
    public ASLFace(int i0, int i1, int i2) { v0 = i0; v1 = i1; v2 = v3 = i2; }
    public ASLFace(int i0, int i1, int i2, int i3) { v0 = i0; v1 = i1; v2 = i2;  v3 = i3; }
    public ASLFace Clone() { return new ASLFace(v0, v1, v2, v3); }
}

[Serializable]
public class ASLMesh
{
    public string Name;
    public ASLMaterial Material;
    public bool HasNormals = false, HasTextures = false;
    public ASLStack<ASLVertex> Vertices = new ASLStack<ASLVertex>(10, 10);
    public ASLStack<ASLFace> Faces = new ASLStack<ASLFace>(10, 10);
    public ASLCGGroup Group;
    public ASLCGLayer Layer;
    public ASLStack<ASLMesh> SubMeshes = null;
```

```
        public ASLMesh() { }
        public ASLMesh(ASLMaterial MM) { Material = MM; }
        public ASLMesh(string name, string layername, string matname, int R, int G, int B)
        {
            Name = name;
            ASLColour col = new ASLColour(R, G, B);
            Material = new ASLMaterial(matname, col, col, new ASLColour(255, 255, 255), 0, 0);
            Layer = new ASLCGLayer( layername);
        }

        public ASLMesh Clone()
        {
            ASLMesh M = new ASLMesh();
            if (Material != null) M.Material = Material.Clone();
            M.HasNormals = HasNormals; M.HasTextures = HasTextures;
            M.Group = Group; M.Layer = Layer;
            foreach (ASLVertex v in Vertices) M.AddV(v.Clone());
            foreach (ASLFace f in Faces) M.AddF(f.Clone());
            if (SubMeshes != null) foreach (ASLMesh m in SubMeshes) M.AddMesh(m.Clone());
            return M;
        }

        /// <summary>
        /// Adds a Vertex
        /// </summary>
        /// <param name="v"></param>
        /// <returns></returns>
        public int AddV(ASLVertex v) { return Vertices.Add(v); }

        public int AddV(ASL3d p) { return Vertices.Add(new ASLVertex(p)); }
        public int AddV(ASL2d p) {return Vertices.Add(new ASLVertex(new ASL3d(p))); }

        /// <summary>
        /// Adds a Face
        /// </summary>
        /// <param name="f"></param>
        /// <returns></returns>
        public int AddF(ASLFace f) { return Faces.Add(f); }

        public int AddF(int i0, int i1, int i2) { return Faces.Add(new ASLFace(i0, i1, i2)); }
        public int AddF(int i0, int i1, int i2, int i3) {return Faces.Add(new ASLFace(i0,i1, i2, i3))   ;}
        /// <summary>
        ///Adds a subsidiary mesh to this one. AddMesh will initialize the SubMeshes list automatically if this is the first to be
added
        /// </summary>
        /// <param name="M"></param>
        /// <returns></returns>
        public int AddMesh(ASLMesh M)
        {
            if (SubMeshes == null) SubMeshes = new ASLStack<ASLMesh>(5, 5);
            return SubMeshes.Add(M);
        }

        /// <summary>
        /// Shifts the entire mesh by an amount v.x in the x direction, v.y in the y and v.z in the z
        /// </summary>
        /// <param name="v">The translation vector [v.x, v.y, v.z]</param>
```

```
public void Translate(ASL3d v)
{
   foreach (ASLVertex vtx in Vertices) vtx.Vertex += v;
   if (SubMeshes != null) foreach (ASLMesh m in SubMeshes) m.Translate(v);
}

/// <summary>
/// Mirrors (reflects) the mesh in the given Plane
/// </summary>
/// <param name="pp">The mirroring Plane</param>
public void Mirror(ASLPlane pp)
{
   foreach (ASLVertex vtx in Vertices)
   {
      vtx.Vertex = pp.Mirror(new ASL3dPoint(vtx.Vertex));
      if (HasNormals) vtx.Normal = pp.Mirror(new ASL3dVector(vtx.Normal));
      if (SubMeshes != null) foreach (ASLMesh m in SubMeshes) m.Mirror(pp);
   }
}

/// <summary>
/// Rotates the entire mesh through an angle th about a line in the direction Axis through a point Centre
/// </summary>
/// <param name="Centre">A point on the Axis of rotation</param>
/// <param name="Axis">The direction of the Axis of Rotation</param>
/// <param name="th">The Angle through which to Rotate the Mesh</param>
public void Rotate(ASL3dPoint Centre, ASL3dVector Axis, ASLAngle th)
{
   ASLRotor Q = new ASLRotor(th, Axis);
   foreach (ASLVertex vtx in Vertices)
   {
      vtx.Vertex = (ASL3d)Q.Rotate(Centre, (ASL3dPoint)vtx.Vertex);
      if (HasNormals) vtx.Normal = (ASL3d)Q.Rotate((ASL3dVector)vtx.Normal);
   }
   if (SubMeshes != null) foreach (ASLMesh m in SubMeshes) m.Rotate(Centre, Axis, th);
}
/// <summary>
/// Rotates the entire mesh through an angle th about the given Line
/// </summary>
/// <param name="L">The Axis of rotation</param>
/// <param name="th">The angle of rotation</param>
public void Rotate(ASLLine L, ASLAngle th) { Rotate(L.P0, L.V, th); }

/// <summary>
/// Rotates the entire mesh in the XY plane about a given Centre and through a given angle. Only the x and y
/// coordinates will be affected by the rotation
/// </summary>
/// <param name="Centre">The Center of rotation. Its z coordinate is ignored</param>
/// <param name="th">The angle of rotation</param>
public void RotateXY(ASL3d Centre, ASLAngle th)
{
   foreach (ASLVertex vtx in Vertices)
   {
      double dx = vtx.Vertex.x - Centre.x, dy = vtx.Vertex.y - Centre.y;
      vtx.Vertex.x = Centre.x + dx * th.x - dy * th.y;
      vtx.Vertex.y = Centre.y + dx * th.y + dy * th.x;
      if (HasNormals)
      {
```

```
            vtx.Normal.x = vtx.Normal.x * th.x - vtx.Normal.y * th.y;
            vtx.Normal.y = vtx.Normal.x * th.y + vtx.Normal.y * th.x;
        }
    }
    if (SubMeshes != null) foreach (ASLMesh m in SubMeshes) m.RotateXY(Centre, th);
}


public void RotateXZ(ASL3d Centre, ASLAngle th)
{
    foreach (ASLVertex vtx in Vertices)
    {
        double dx = vtx.Vertex.x - Centre.x, dz = vtx.Vertex.z - Centre.z;
        vtx.Vertex.x = Centre.x + dx * th.x - dz * th.y;
        vtx.Vertex.z = Centre.z + dx * th.y + dz * th.x;
        if (HasNormals)
        {
            vtx.Normal.x = vtx.Normal.x * th.x - vtx.Normal.z * th.y;
            vtx.Normal.z = vtx.Normal.x * th.y + vtx.Normal.z * th.x;
        }
    }
    if (SubMeshes != null) foreach (ASLMesh m in SubMeshes) m.RotateXZ(Centre, th);
}

public void RotateYZ(ASL3d Centre, ASLAngle th)
{
    foreach (ASLVertex vtx in Vertices)
    {
        double dz = vtx.Vertex.z - Centre.z, dy = vtx.Vertex.y - Centre.y;
        vtx.Vertex.x = Centre.x + dy * th.x - dz * th.y;
        vtx.Vertex.y = Centre.y + dy * th.y + dz * th.x;
        if (HasNormals)
        {
            vtx.Normal.x = vtx.Normal.y * th.x - vtx.Normal.z * th.y;
            vtx.Normal.y = vtx.Normal.y * th.y + vtx.Normal.z * th.x;
        }
    }
    if (SubMeshes != null) foreach (ASLMesh m in SubMeshes) m.RotateYZ(Centre, th);
}

/// <summary>
/// Scales the entire mesh about a Centre point by the amounts given as the x, y and z coordinates of Scale
/// </summary>
/// <param name="Centre">The Centre about which the mesh is scaled</param>
/// <param name="Scale">Scale.x gives the x scaling factor, Scale.y the y and Scale.z the z</param>
public void Scale(ASL3d Centre, ASL3d Scale)
{
    foreach (ASLVertex vtx in Vertices)
    {
        vtx.Vertex.x = Scale.x * (vtx.Vertex.x - Centre.x);
        vtx.Vertex.y = Scale.y * (vtx.Vertex.y - Centre.y);
        vtx.Vertex.z = Scale.z * (vtx.Vertex.z - Centre.z);
        if (HasTextures)
        {
            vtx.Texture.x = Scale.x * (vtx.Texture.x - Centre.x);
            vtx.Texture.y = Scale.y * (vtx.Texture.y - Centre.y);
            //very unsure about this
        }
```

```
        }
    }
    public void RotateXY(ASL3dPoint Centre, ASLAngle th) { RotateXY((ASL3d)Centre, th);}
    public void RotateYZ(ASL3dPoint Centre, ASLAngle th){ RotateYZ((ASL3d)Centre, th);}
    public void RotateXZ(ASL3dPoint Centre, ASLAngle th){ RotateXZ((ASL3d)Centre, th);}
    public void Scale(ASL3dPoint Centre, ASL3d scale) { Scale((ASL3d)Centre, scale); }

    public void GetMinMax(ref ASL3d min, ref ASL3d max, int level)
    {
        if (level == 0) { min.x = min.y = min.z = double.PositiveInfinity; max.x = max.y = max.z = double.NegativeInfinity; }
        foreach (ASLVertex v in Vertices)
        {
            if (v.Vertex.x < min.x) min.x = v.Vertex.x; if (v.Vertex.x > max.x) max.x = v.Vertex.x;
            if (v.Vertex.y < min.y) min.y = v.Vertex.y; if (v.Vertex.y > max.y) max.y = v.Vertex.y;
            if (v.Vertex.z < min.z) min.z = v.Vertex.z; if (v.Vertex.z > max.z) max.z = v.Vertex.z;
        }
        if (SubMeshes != null) foreach (ASLMesh M in SubMeshes) M.GetMinMax(ref min, ref max, level + 1);
    }

    public ASLStack<ASLMeshEdge> OuterEdges()
    {
        ASLStack<ASLMeshEdge> Edges = new ASLStack<ASLMeshEdge>(Faces.Count + Vertices.Count, 10);
        foreach (ASLFace F in Faces)
        {
            Edges.Add(new ASLMeshEdge(F.v0, F.v1));
            Edges.Add(new ASLMeshEdge(F.v1, F.v2));
            if (F.v2 == F.v3) Edges.Add(new ASLMeshEdge(F.v2, F.v0)); else { Edges.Add(new ASLMeshEdge(F.v2, F.v3));
Edges.Add(new ASLMeshEdge(F.v3, F.v0)); }
        }
        for (int ii = 1; ii < Edges.Count; ii++)
        {
            if (Edges[ii].IsMatched) continue;
            for (int jj = ii + 1; jj <= Edges.Count; jj++)
            {
                if (Edges[jj].IsMatched) continue;
                if (Edges[ii] == Edges[jj]) { Edges[ii].IsMatched = true; Edges[jj].IsMatched = true; }
            }
        }
        ASLStack<ASLMeshEdge> Outers = new ASLStack<ASLMeshEdge>(Edges.Count, 10);
        for (int ii = 1; ii <= Edges.Count; ii++) if (!Edges[ii].IsMatched) Outers.Add(Edges[ii]);
        return Outers;
    }
}
```