Lecture 09

- Re-constructing the connection of an STL model
- Lattice-Line Intersection

Re-constructing connection of an STL model

Problems of an STL model.

- No connection information.
- No grouping information.
- Nothing!

STL is probably THE WORST possible data format of a 3D model.

But, as usual, the worst thing becomes a standard.

Re-constructing connection of an STL model

- To do something useful, first thing to do is to re-construct connections.
- Need to convert a set of disconnected triangles to a list of vertices + a list of polygons.
- A polygon must be represented as a chain of vertices, not raw 3D coordinates.

Introducing YsShellExt class

- A polygonal mesh class.
- Can store a set of:
  - Vertices
  - Polygons
  - Constraint edges (feature edges)
  - Face Groups
  - Volumes
- In this example, we only use vertices and polygons.
- Vertices and polygons are accessed by:
  - YsShell::VertexHandle
  - YsShell::PolygonHandle
- YsShell is a base class of YsShellExt.

Introducing YsShellExt class

Problem:

- Transfer triangles of the STL model in std::vector <float> to a YsShellExt object.
- To add a triangle, you also need to add vertices.
- However, you don’t want to create multiple vertices taking the same position.

Strategy:

- When you add a triangle, for each corner,
  – Search a vertex that is already created at the corner.
  – If such a vertex is found, use it.
  – If not found, create a new vertex.

- First make a raw two-way conversions between YsShellExt and vertex and normal arrays.
- Then, modify the conversion from vertex and normal arrays to YsShellExt so that no duplicate vertex is created.

Converting an STL model into a YsShellExt object as is:

1. Copy project from picking project. Rename TARGET_NAME to glsl3d_stl_to_ysshell
2. Add geblkernel to LIBRARY_DEPENDENCY:
3. Add:

```
#include <ysshellext.h>
```

4. Add two member functions:

```
static void VtxNomToYsShell(
   YsShellExt &shl,
   const std::vector <float> &vtx,const std::vector <float> &nom);
static void YsShellToVtxNom(
   std::vector <float> &vtx,std::vector <float> &nom,
   const YsShellExt &shl);
```

```
/* static */ void FsLazyWindowApplication::VtxNomToYsShell(
    YsShellExt &shl,const std::vector <float> &vtx,const std::vector <float> &nom)
{
    shl.CleanUp();
    for(int i=0; i<vtx.size()/9; ++i)
    {
        const YsVec3 nom(nom[i*9],nom[i*9+1],nom[i*9+2]);
        const YsVec3 vtPos[3]=
        {
            YsVec3(vtx[i*9  ],vtx[i*9+1],vtx[i*9+2]),
            YsVec3(vtx[i*9+3],vtx[i*9+4],vtx[i*9+5]),
            YsVec3(vtx[i*9+6],vtx[i*9+7],vtx[i*9+8]),
        };
        YsShell::VertexHandle vtHd[3];
        vtHd[0]=shl.AddVertex(vtPos[0]);
        vtHd[1]=shl.AddVertex(vtPos[1]);
        vtHd[2]=shl.AddVertex(vtPos[2]);
        YsShell::PolygonHandle plHd;
        plHd=shl.AddPolygon(3,vtHd);
        shl.SetPolygonNormal(plHd,nom);
    }
}
```

AddVertex function adds a new vertex and returns a vertex handle

AddPolygon function adds a new polygon with the given array of vertex handles and returns a polygon handle

SetPolygonNormal function assigns a normal vector to the polygon.

```
/* static */ void FsLazyWindowApplication::YsShellToVtxNom(
    std::vector <float> &vtx,
    std::vector <float> &nom,
    std::vector <float> &col,
    const YsShellExt &shl)
{
    vtx.clear();
    nom.clear();
    col.clear();
    for(auto plHd : shl.AllPolygon())
    {
        auto plVtHd=shl.GetPolygonVertex(plHd);
        if(3<=plVtHd.GetN())
        {
            auto plNom=shl.GetNormal(plHd);
            for(auto vtHd : plVtHd)
            {
                auto vtPos=shl.GetVertexPosition(vtHd);
                vtx.push_back(vtPos.xf());
                vtx.push_back(vtPos.yf());
                vtx.push_back(vtPos.zf());
                nom.push_back(plNom.xf());
                nom.push_back(plNom.yf());
                nom.push_back(plNom.zf());
                col.push_back(0);
                col.push_back(0);
                col.push_back(1);
                col.push_back(1);
            }
        }
    }
}
```

GetPolygonNormal function takes a polygon handle, and returns a YsArray of vertex handles.

GetNormal function returns a normal vector assigned to a polygon.

GetVertexPosition function returns a position of the vertex.

5. Add a member variable:

```
YsShellExt shl;
```

6. Modify LoadBinaryStl function as:

```
void FsLazyWindowApplication::LoadBinaryStl(const char fn[])
{
    LoadBinaryStl(vtx,nom,fn);
    col.clear();
    for(int i=0; i<vtx.size()/3; ++i)
    {
        col.push_back(0);
        col.push_back(0);
        col.push_back(1);
        col.push_back(1);
    }
    VtxNomToYsShell(shl,vtx,nom);
    CacheBoundingBox();
}
```

YsShellExt class has its own LoadStl, but let's not use it this time yet.

7. Modify CacheBoundingBox

```
void FsLazyWindowApplication::CacheBoundingBox(void)
{
    shl.GetBoundingBox(min,max);
}
```

8. Modify PickedTriangle and PickedPoint functions as:

```
YsShell::PolygonHandle FsLazyWindowApplication::PickedTriangle(int mx,int my) const
{
    YsVec3 o,v;
    drawEnv.TransformScreenCoordTo3DLine(o,v,mx,my);

    YsShell::PolygonHandle picked=nullptr;
    double pickedDist=0.0;
    for(auto plHd : shl.AllPolygon())
    {
        auto plVtHd=shl.GetPolygonVertex(plHd);
        const YsVec3 tri[3]=
        {
            shl.GetVertexPosition(plVtHd[0]),
            shl.GetVertexPosition(plVtHd[1]),
            shl.GetVertexPosition(plVtHd[2]),
        };
        YsPlane pln;
        pln.MakePlaneFromTriangle(tri[0],tri[1],tri[2]);

        YsVec3 itsc;
        if(YSOK==pln.GetIntersection(itsc,o,v))
        {
            auto side=YsCheckInsideTriangle3(itsc,tri);
            if(YSINSIDE==side || YSBOUNDARY==side)
            {
                auto dist=(itsc-o)*v; // Gives distance
                if(0.0<dist && (picked==nullptr || dist<pickedDist))
                {
                    picked=plHd;
                    pickedDist=dist;
                }
            }
        }
    }

    return picked;
}
```

```
YsShell::VertexHandle FsLazyWindowApplication::PickedVertex(int mx,int my) const
{
    int wid,hei;
    FsGetWindowSize(wid,hei);

    double pickedZ=YsInfinity;
    YsShell::VertexHandle pickedVtHd=nullptr;
    for(auto vtHd : shl.AllVertex())
    {
        YsVec3 pos=shl.GetVertexPosition(vtHd);
        drawEnv.GetViewMatrix().Mul(pos,pos,1.0);
        drawEnv.GetProjectionMatrix().Mul(pos,pos,1.0);
        if(-1.0<=pos.z() && pos.z()<=1.0)
        {
            const double u=(pos.x()+1.0)/2.0;
            const double v=(pos.y()+1.0)/2.0;

            int x=(int)((double)wid*u);
            int y=hei-(int)((double)hei*v);
            if(mx-8<=x && x<=mx+8 && my-8<=y && y<=my+8)
            {
                if(nullptr==pickedVtHd || pos.z()<pickedZ)
                {
                    pickedVtHd=vtHd;
                    pickedZ=pos.z();
                }
            }
        }
    }

    return pickedVtHd;
}
```

9. Let’s lift a polygon instead of changing color this time.

```
if(evt==FSMOUSEEVENT_LBUTTONDOWN)
{
    drawEnv.SetWindowSize(wid,hei);
    drawEnv.SetViewportByTwoCorner(0,0,wid,hei);
    drawEnv.TransformScreenCoordTo3DLine(lastClick[0],lastClick[1],mx,my);
    lastClick[1]*=80.0;
    lastClick[1]+=lastClick[0];

    auto plHd=PickedTriangle(mx,my);
    if(nullptr!=plHd)
    {
        auto plVtHd=shl.GetPolygonVertex(plHd);
        auto plNom=shl.GetNormal(plHd);
        for(auto vtHd : plVtHd)
        {
            auto pos=shl.GetVertexPosition(vtHd);
            shl.SetVertexPosition(vtHd,pos+plNom);
        }
        YsShellToVtxNom(vtx,nom,col,shl);
    }
}
if(evt==FSMOUSEEVENT_RBUTTONDOWN)
{
}
```

- Clicked polygon gets disconnected from the other polygons because there is no connection.
- Need to recover connection.

O(N^2) method

```
void FsLazyWindowApplication::VtxNomToYsShell(
    YsShellExt &shl,const std::vector <float> &vtx,const std::vector <float> &nom)
{
    shl.CleanUp();
    for(int i=0; i<vtx.size()/9; ++i)
    {
        const YsVec3 nom(nom[i*9],nom[i*9+1],nom[i*9+2]);
        const YsVec3 vtPos[3]=
        {
            YsVec3(vtx[i*9  ],vtx[i*9+1],vtx[i*9+2]),
            YsVec3(vtx[i*9+3],vtx[i*9+4],vtx[i*9+5]),
            YsVec3(vtx[i*9+6],vtx[i*9+7],vtx[i*9+8]),
        };
        YsShell::VertexHandle vtHd[3];
        for(int i=0; i<3; ++i)
        {
            vtHd[i]=nullptr;
            for(auto tstVtHd : shl.AllVertex())
            {
                if(shl.GetVertexPosition(tstVtHd)==vtPos[i])
                {
                    vtHd[i]=tstVtHd;
                    break;
                }
            }
            if(nullptr==vtHd[i])
            {
                vtHd[i]=shl.AddVertex(vtPos[i]);
            }
        }
        YsShell::PolygonHandle plHd;
        plHd=shl.AddPolygon(3,vtHd);
        shl.SetPolygonNormal(plHd,nom);
    }
}
```

For each triangle corner, check if a vertex at that location is already in the YsShellExt.

Easy to implement.

Good for small STLs, but bad for large STLs.

- It can easily be made close to to O(N) assuming that the vertices are uniformly distributed.
- Since vertices may not be distributed uniformly, it may get slower than expectation, but typically gives a good speed boost.

1. Prepare a lattice that covers the entire domain with a reasonably good resolution.
2. For each polygon vertex, check if a vertex is already created in YsSellExt before creating one. This check can be done very quickly by the Lattice.
3. If a already-created vertex is not found, create one, and register it in the Lattice.

1. Copy and include lattice.h
2. Add GetBoundingBox function:

```
void FsLazyWindowApplication::GetBoundingBox(
    YsVec3 &min,YsVec3 &max,const std::vector <float> &vtx)
{
    auto nVtx=vtx.size()/3;
    if(0==nVtx)
    {
        min=YsVec3::Origin();
        max=YsVec3::Origin();
    }
    else
    {
        YsBoundingBoxMaker3 mkBbx;
        for(decltype(nVtx) i=0; i<nVtx; ++i)
        {
            YsVec3 pos(vtx[i*3],vtx[i*3+1],vtx[i*3+2]);
            mkBbx.Add(pos);
        }
        mkBbx.Get(min,max);
    }
}
```

3. Add PrepareLattice function:

```
void FsLazyWindowApplication::PrepareLatticeForConnection(
    Lattice3d <std::vector <YsShell::VertexHandle> > &ltc,const std::vector <float> &vtx)
{
    YsVec3 min,max;
    GetBoundingBox(min,max,vtx);
    double d=(max-min).GetLength()/100.0;
    min-=YsXYZ()*d; // Make absolutely sure that all vertices are inside.
    max+=YsXYZ()*d;

    auto nVtx=vtx.size()/3;
    ltc.Create(100,100,100);
    ltc.SetDimension(min,max);
}
```

4. Modify VtxNomToYsShell

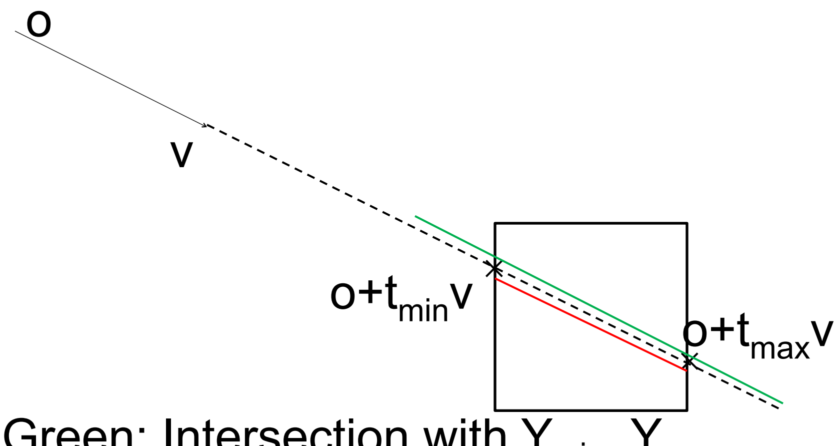
```
void FsLazyWindowApplication::VtxNomToYsShell(
    YsShellExt &shl,const std::vector <float> &vtx,const std::vector <float> &nom)
{
    Lattice3d <std::vector <YsShell::VertexHandle> > ltc;
    PrepareLatticeForConnection(ltc,vtx);
    shl.CleanUp();
    for(int i=0; i<vtx.size()/9; ++i)
    {
        const YsVec3 plNom(nom[i*9],nom[i*9+1],nom[i*9+2]);
        const YsVec3 vtPos[3]=
        {
            YsVec3(vtx[i*9  ],vtx[i*9+1],vtx[i*9+2]),
            YsVec3(vtx[i*9+3],vtx[i*9+4],vtx[i*9+5]),
            YsVec3(vtx[i*9+6],vtx[i*9+7],vtx[i*9+8]),
        };
        YsShell::VertexHandle vtHd[3];
        for(int i=0; i<3; ++i)
        {
            vtHd[i]=nullptr;
            auto idx=ltc.GetBlockIndex(vtPos[i]);
            if(true==ltc.IsInRange(idx))
            {
                for(auto tstVtHd : ltc.Elem(idx.x(),idx.y(),idx.z()))
                {
                    if(shl.GetVertexPosition(tstVtHd)==vtPos[i])
                    {
                        vtHd[i]=tstVtHd;
                        break;
                    }
                }
            }
            if(nullptr==vtHd[i])
            {
                vtHd[i]=shl.AddVertex(vtPos[i]);
                if(true==ltc.IsInRange(idx))
                {
                    ltc.Elem(idx.x(),idx.y(),idx.z()).push_back(vtHd[i]);
                }
            }
        }
        YsShell::PolygonHandle plHd;
        plHd=shl.AddPolygon(3,vtHd);
        shl.SetPolygonNormal(plHd,plNom);
    }
}
```

Lattice & Line Intersection

- It is easy to register a 3D primitive to a lattice.
- It can be used for accelerated intersection check such as picking.
- What we need is a way to find cells that intersect with a line.

Clipping

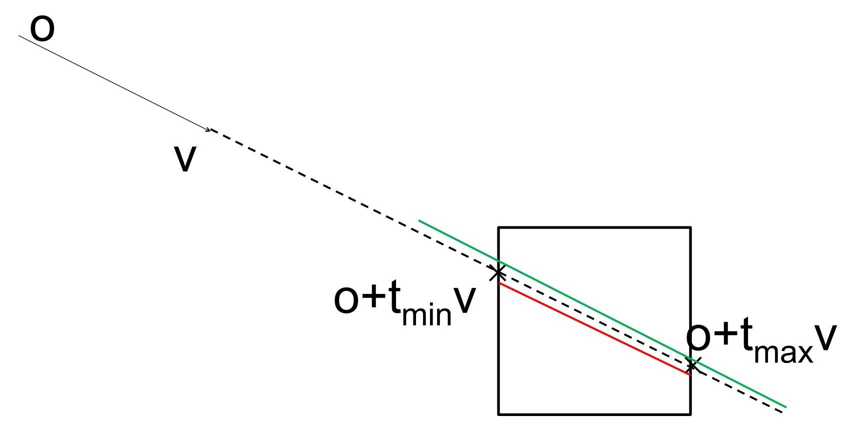
- First information we need is two points where the line intersects with the outer box of the lattice.
- Calculate intersection with $x=X_{min}$, $x=X_{max}$, $y=Y_{min}$, $y=Y_{max}$, $z=Z_{min}$, $z=Z_{max}$, and take the overlapping segment.



Green: Intersection with Y_{min}, Y_{max}

Red: Intersection with X_{min}, X_{max}

- Initialize t_{min}=-∞ and t_{max} = ∞.
- For each dimension, calculate and update t_{min} and t_{max}.
- The condition to have a clipped segment is:
  - After doing it for all three dimensions, $t_{min} < t_{max}$
  - If the coordinate of the two intersecting points are (x0,y0,z0), (x1,y1,z1),
    - minx<=x0<=maxx
    - miny<=y0<=maxy
    - minz<=z0<=maxz



Green: Intersection with Y_{min}, Y_{max}

Red: Intersection with X_{min}, X_{max}

Check clipping

1. Copy stl_to_ysshell project and make TARGET_NAME as glsl3d_lattice_line
2. In lattice.h, add:

   ```
   #include <math.h>
   ```

   and ClipLine function.

```
template <class T>
bool Lattice3d<T>::ClipLine(
    YsVec3 &p0,YsVec3 &p1,const YsVec3 o,const YsVec3 v,const YsVec3 &min,const YsVec3 &max) const
{
    // YsTolerance is a small number
    // YsInfinity is a very large number.
    double tmin=-YsInfinity,tmax=YsInfinity;

    for(int dim=0; dim<3; ++dim)
    {
        if(YsTolerance<fabs(v[dim]))
        {
            double t0=(min[dim]-o[dim])/v[dim];
            double t1=(max[dim]-o[dim])/v[dim];
            YsVec3 i0=o+t0*v;
            YsVec3 i1=o+t1*v;
            // Reduce adverse effect from numerical error.
            // minx may not be equal to
            // ox+vx*((minx-ox)/vx)
            i0[dim]=min[dim];
            i1[dim]=max[dim];

            if(t0>t1)
            {
                std::swap(t0,t1);
                std::swap(i0,i1);
            }

            if(tmin<t0)
            {
                tmin=t0;
                p0=i0;
            }
            if(t1<tmax)
            {
                tmax=t1;
                p1=i1;
            }
        }
    }

    if(tmax<tmin)
    {
        return false;
    }

    for(int dim=0; dim<3; ++dim)
    {
        if((p0[dim]<min[dim] && p1[dim]<min[dim]) ||
           (p0[dim]>max[dim] && p1[dim]>max[dim]))
        {
            return false;
        }
    }

    return true;
}
```

v[0], v[1], and v[2] give x,y, and z components respectively.

```
template <class T>
bool Lattice3d<T>::ClipLine(YsVec3 &p1,YsVec3 &p2,const YsVec3 o,const YsVec3 n) const
{
    return ClipLine(p1,p2,o,n,min,max);
}
```

3. Make Lattice a member variable (move from VtxNomToYsShell), and Make VtxNomToYsShell a non-static function (because it populates a lattice).
4. Modify FSMOUSEEVENT_LBUTTONDOWN handling as:

```
if(evt==FSMOUSEEVENT_LBUTTONDOWN)
{
   drawEnv.SetWindowSize(wid,hei);
   drawEnv.SetViewportByTwoCorner(0,0,wid,hei);
   drawEnv.TransformScreenCoordTo3DLine(lastClick[0],lastClick[1],mx,my);
   lastClick[1]*=80.0;
   lastClick[1]+=lastClick[0];

   if(true!=ltc.ClipLine(lastClick[0],lastClick[1],lastClick[0],lastClick[1]-lastClick[0]))
   {
      printf("No intersection.\n");
   }
}
```

5. Change polygon color to 0,0,0,0.5
6. In Draw, draw line (lastClick[0]-lastClick[1]) first.

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
```

Then, the line will be clipped by the lattice outer box.

Finding intersecting cells.

- After finding intersection p0,p1, find the minimum and maximum cell indices that encloses p0 and p1, and then for each of those cells check if the cell intersects with the line.

1. In lattice.h add:

   ```
   #include <vector>
   ```

2. Add two member functions in Lattice3d

   ```
   void GetBlockRange(YsVec3 &min,YsVec3 &max,YsVec3i idx) const;
   std::vector <YsVec3i> GetIntersectingBlock(const YsVec3 o,const YsVec3 n) const;
   ```

```
template <class T>
void Lattice3d<T>::GetBlockRange(YsVec3 &min,YsVec3 &max,YsVec3i idx) const
{
    const double dx=dgn.x()/(double)nx;
    const double dy=dgn.y()/(double)ny;
    const double dz=dgn.z()/(double)nz;
    min.SetX(this->min.x()+dx*(double)idx.x());
    min.SetY(this->min.y()+dy*(double)idx.y());
    min.SetZ(this->min.z()+dz*(double)idx.z());
    max=min;
    max.AddX(dx);
    max.AddY(dy);
    max.AddZ(dz);
}
```

```
template <class T>
std::vector <YsVec3i> Lattice3d<T>::GetIntersectingBlock(const YsVec3 o,const YsVec3 v) const
{
    std::vector <YsVec3i> itscIdx;
    YsVec3 p0,p1;
    if(true==ClipLine(p0,p1,o,v))
    {
        YsVec3i i0=GetBlockIndex(p0);
        YsVec3i i1=GetBlockIndex(p1);
        YsVec3i iMin,iMax;
        iMin.Set(
            YsSmaller(i0.x(),i1.x()),
            YsSmaller(i0.y(),i1.y()),
            YsSmaller(i0.z(),i1.z()));
        iMax.Set(
            YsGreater(i0.x(),i1.x()),
            YsGreater(i0.y(),i1.y()),
            YsGreater(i0.z(),i1.z()));
        for(auto i : YsVec3iRange(iMin,iMax))
        {
            YsVec3 min,max,t0,t1;
            GetBlockRange(min,max,i);
            if(true==ClipLine(t0,t1,o,v,min,max))
            {
                itscIdx.push_back(i);
            }
        }
    }
    return itscIdx;
}
```

Can it be faster?

- Yes!
- Find one cell that intersects with the line, and the start a flood-fill algorithm to find all intersecting cells.

- I let you think by yourself.