Lecture 23

Password: fishAndChips

- No class 4/20 (Wed)
- Spend time for finishing up the project.

Debugging Challenge

- The following code is a typical flood-fill function.
- What's the error?
  - In what case does this flood-fill fail?
  - How would you fix it?

```
class Vec2
{
public:
    int x,y;
};
void FloodFill(char ltc[],int ltcWid,int ltcHei,int x0,int y0,char from,char to)
{
    if(x0<0 || ltcWid<=x0 || y0<0 || ltcHei<=y0 || ltc[y0*ltcWid+x0]!=from)
    {
        return;
    }
    std::vector <Vec2> todo;
    Vec2 vec2;
    vec2.x=x0;
    vec2.y=y0;
    todo.push_back(vec2);
    while(0<todo.size())
    {
        auto pos=todo.back();
        todo.pop_back();
        Vec2 nei[4];
        nei[0].x=pos.x-1;        nei[0].y=pos.y;
        nei[1].x=pos.x+1;        nei[1].y=pos.y;
        nei[2].x=pos.x;          nei[2].y=pos.y-1;
        nei[3].x=pos.x;          nei[3].y=pos.y+1;
        for(auto n : nei)
        {
            if(0<=n.x && n.x<ltcWid && 0<=n.y && n.y<ltcHei &&
              ltc[n.y*ltcWid+n.x]==from)
            {
                ltc[n.y*ltcWid+n.x]=to;
                todo.push_back(n);
            }
        }
    }
}
```

If the first cell is not within the lattice, or already painted, do nothing.

Add the first point (x0,y0) to the todo list.

Four neighbors

There is an error somewhere!

If the neighbor needs to be painted, do so, and add to todo list.

- Creating a dialog
- Data structure for undo- and redo-ing
- Edge-Collapsing
- Adding Undo/Redo
- Edge-Swapping

Creating a dialog

- Common widgets:
  - Button
    - Push Button
    - Radio Button
    - Check Box
  - Text Box (Edit Control. This FsGuiLib supports single-line edit only at this time.)
  - List Box
  - Drop List
  - Combo Box (Not supported in this toolkit, but common in many other toolkits.)
  - Number Box
  - Color Palette
  - Slider
  - Tree Control

Creating a dialog

- In this toolkit, the events are directed by the virtual-function method.
- Virtual-function method was not appropriate for the menus, but is it good for the dialogs?

Creating a dialog

- First you need to create widgets on the dialog.
  - In MFC, ClassWizard.
  - In Qt, Qt Designer.
  - In XCode, Interface Builder.
- You can graphically place widgets, but
- if you do it outside C++, you get a code-maintenance problem.
- Also your program is tied to a specific dialog-building tool.
- FsGuiLib does it in C++ code.

Creating a dialog

- Dialog base class: FsGuiDialog
- You create a sub-class of FsGuiDialog.

Creating a dialog class

```
class MoveDialog : public FsGuiDialog
{
public:
    FsGui3DMainCanvas *canvasPtr;

    FsGuiButton *xPlus,*xMinus;
    FsGuiButton *yPlus,*yMinus;
    FsGuiButton *zPlus,*zMinus;
    FsGuiButton *closeBtn;
    void Make(FsGui3DMainCanvas *canvasPtr);
};
void MoveDialog::Make(FsGui3DMainCanvas *canvasPtr)
{
    this->canvasPtr=canvasPtr;
    xPlus= AddTextButton(0,FSKEY_NULL,FSGUI_PUSHBUTTON,L"+X",YSTRUE);
    xMinus=AddTextButton(0,FSKEY_NULL,FSGUI_PUSHBUTTON,L"-X",YSFALSE);
    yPlus= AddTextButton(0,FSKEY_NULL,FSGUI_PUSHBUTTON,L"+Y",YSTRUE);
    yMinus=AddTextButton(0,FSKEY_NULL,FSGUI_PUSHBUTTON,L"-Y",YSFALSE);
    zPlus= AddTextButton(0,FSKEY_NULL,FSGUI_PUSHBUTTON,L"+Z",YSTRUE);
    zMinus=AddTextButton(0,FSKEY_NULL,FSGUI_PUSHBUTTON,L"-Z",YSFALSE);
    closeBtn=AddTextButton(0,FSKEY_NULL,FSGUI_PUSHBUTTON,L"Close",YSTRUE);
    Fit();
}

void FsGui3DMainCanvas::Edit_Move_Dialog(FsGuiPopUpMenuItem *)
{
    auto dlg=FsGuiDialog::CreateSelfDestructiveDialog<MoveDialog>();
    dlg->Make(this);
    AddDialog(dlg);
    ArrangeDialog();
}
```

- You can run at this point and select the new menu to see the dialog is created.
- But, the buttons do nothing since the event-handlers are not defined yet.

Adding a button event handler

```
/* virtual */ void MoveDialog::OnButtonClick(FsGuiButton *btn)
{
    if(btn==closeBtn)
    {
        canvasPtr->RemoveDialog(this);
    }
    else
    {
        YsVec3 move=YsVec3::Origin();
        if(btn==xPlus)
        {
            move.Set(1,0,0);
        }
        else if(btn==xMinus)
        {
            move.Set(-1,0,0);
        }
        else if(btn==yPlus)
        {
            move.Set(0,1,0);
        }
        else if(btn==yMinus)
        {
            move.Set(0,-1,0);
        }
        else if(btn==zPlus)
        {
            move.Set(0,0,1);
        }
        else if(btn==zMinus)
        {
            move.Set(0,0,-1);
        }
        for(auto vtHd : canvasPtr->shl.AllVertex())
        {
            auto pos=canvasPtr->shl.GetVertexPosition(vtHd);
            pos+=move;
            canvasPtr->shl.SetVertexPosition(vtHd,pos);
        }
        canvasPtr->RemakeVertexAttribArray();
        canvasPtr->SetNeedRedraw(YSTRUE);
    }
}
```

Data Structure for Undo- and Redo-ing

- A practical interactive tools have undo- and redo-ing capability.
- Undo-ing allows the user to recover from mistakes.

- How would you implement it? What data structure?

Undo/Redo-ing

- Most primitive implementation: Remembering one step before the last modification.
- Common implementation: Linear undo/redo. Keep track of incremental changes in a linked list.
- Advanced implementation: Design tree. Keep track of incremental changes as a tree structure.

- Commercial CAD packages typically use a design-tree method, and the tree can be saved as a file.

Undo- / Redo-ing

- Editing must be done through limited modifier functions.
- For each modifier, reverse operation must also be defined.
- In other words, every modifier must be written so that the modification can be reversed.
- Example:
  - Displacing a vertex -> Reverse operation moves the vertex to the original location
  - Scaling the model by the factor of *m* -> ?
  - Deleting a polygon -> ?

Undo- / Redo-ing

Common mistakes:

- Reverse operation of scaling by the factor of *m* is not scaling by the factor of 1/*m*. Due to numerical errors, x*m*(1.0/m) may not be equal to x. And how about scaling by zero? Scaling should be implemented as vertex displacement.
- Reverse operation of deleting a polygon may be implemented as creating a polygon. But, how about attributes? Rather, the data structure should be able to temporarily delete and undelete the polygon.

Undo- / Redo-ing in YsShellExtEdit class

- YsShellExtEdit class
  - Sub-class of YsShellExt.
  - Designed for interactive polygonal mesh editing.
  - Also for some algorithms that need trial-and-error approach.
- Each modifier has its own reverse operation.

Undo pointer and redo pointer

- Edit-log is stored as a doubly-linked list of sub-classes of YsShellExtEdit::EditLog.

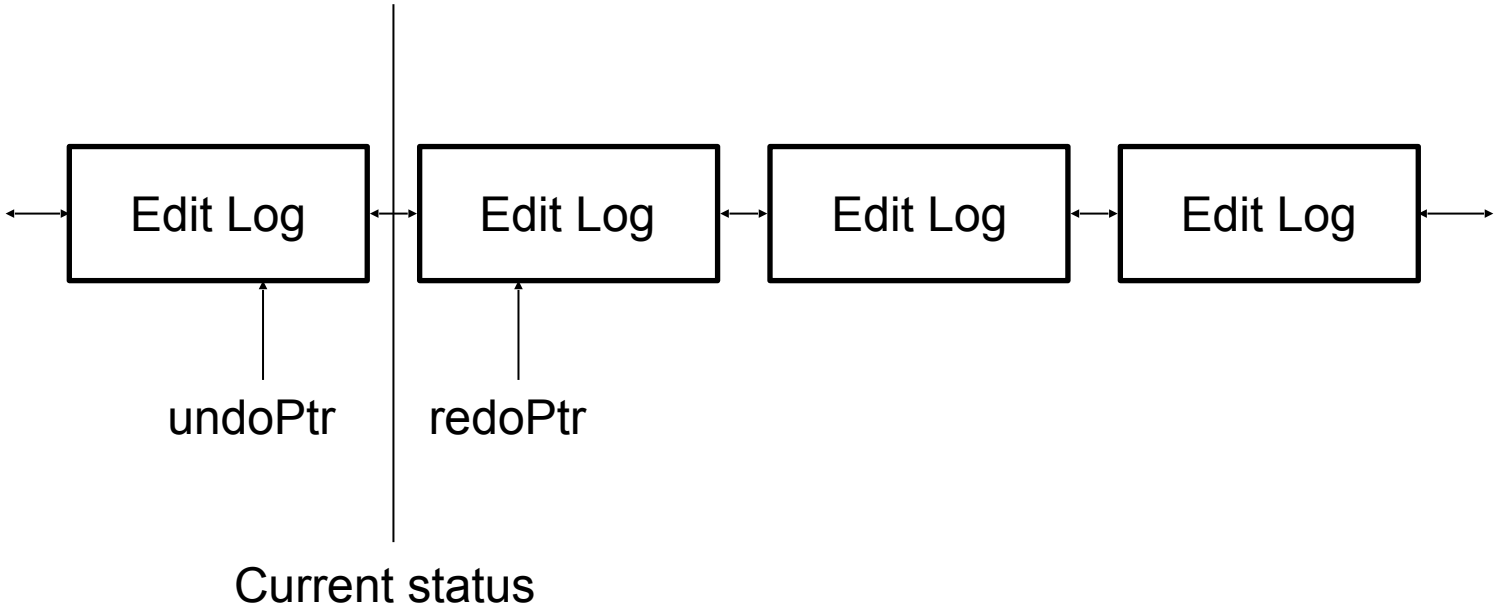
```
class YsShellExtEdit::EditLog
{
public:
    YSBOOL isRememberSelection;
    // Remember Selection is a special undo type.
    // If undoPtr or redoPtr stops at Remember-Selection node, it will go one
    // more step to execute remember selection.

    YSSIZE_T undoCtr;
    EditLog *prev,*next;

    EditLog();
    virtual ~EditLog(){};
    virtual void Undo(YsShellExtEdit *shl)=0;
    virtual void Redo(YsShellExtEdit *shl)=0;
    virtual void NeverBeUndoneAgain(YsShellExtEdit *){};
    virtual void NeverBeRedoneAgain(YsShellExtEdit *){};
};
```

Undo pointer and Redo pointer

- YsShellExtEdit class remembers two pointers:
  - EditLog *undoPtr;
  - EditLog *redoPtr;
- When the user undo-es the previous operation, if nullptr!=undoPtr,
  - undoPtr->Undo();
  - redoPtr=undoPtr;
  - undoPtr=undoPtr->prev;
- Or, redo-es the previously-undone operation, if nullptr!=redoPtr,
  - redoPtr->Redo();
  - undoPtr=redoPtr;
  - redoPtr=redoPtr->next;

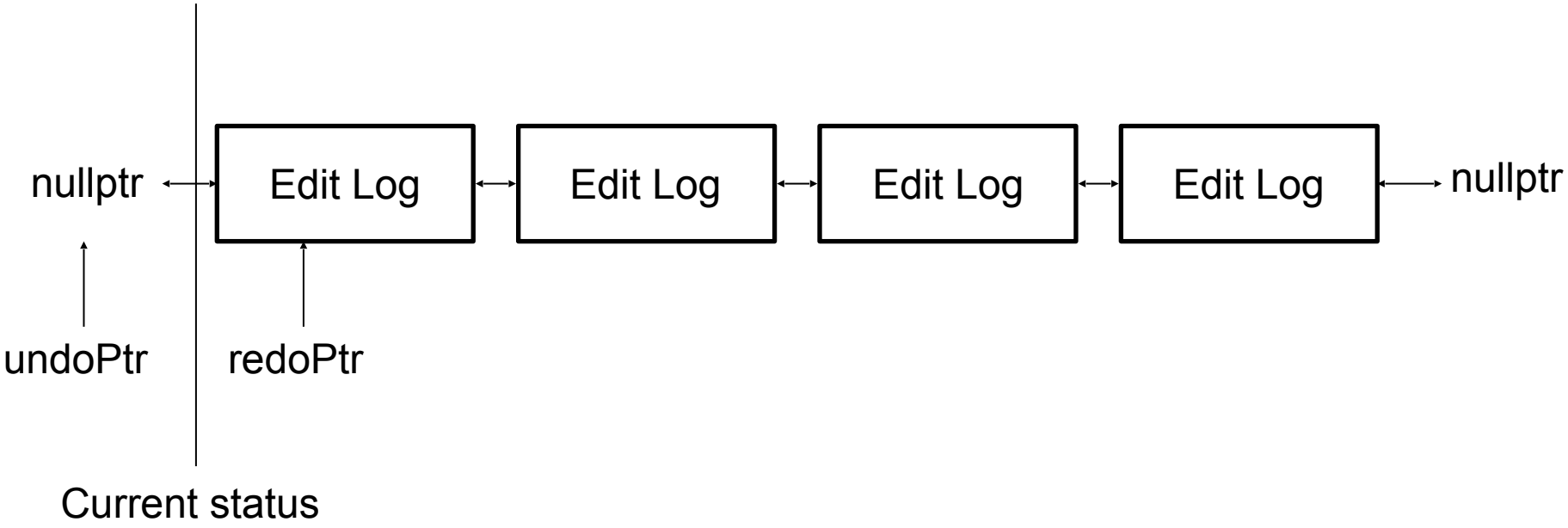
Undo pointer and Redo pointer

- Keeping two pointers is redundant, but make undo-ing and redo-ing code cleaner.



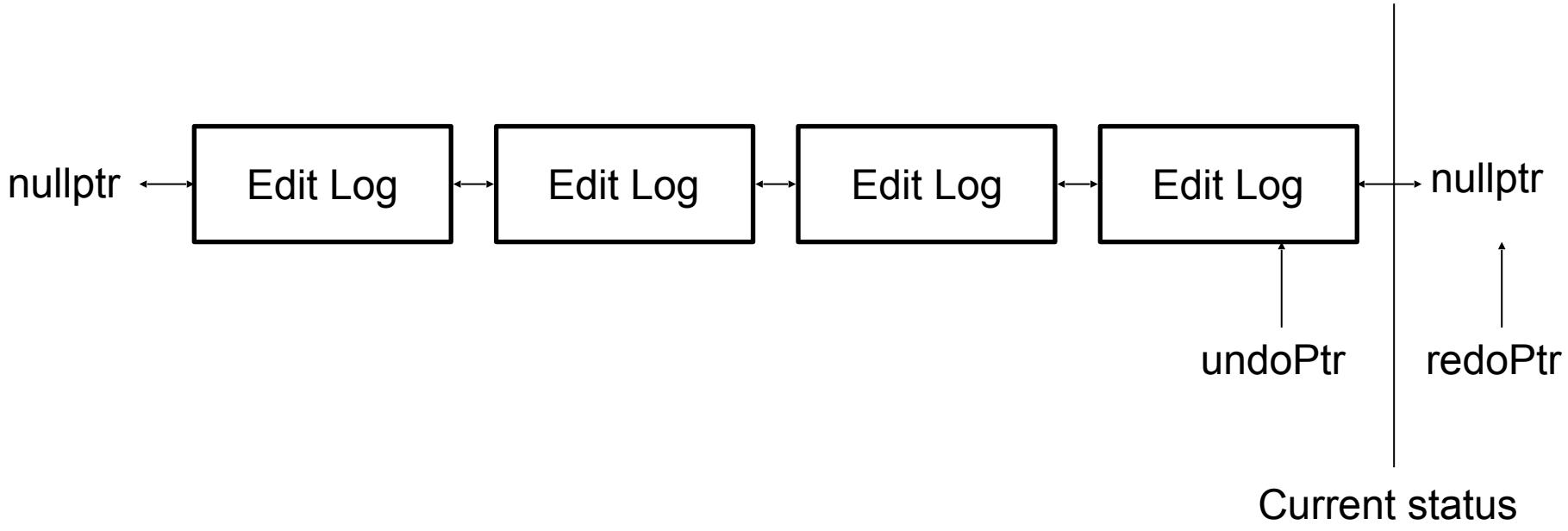
Undo pointer and Redo pointer

- When no more undo-ing is possible, undoPtr is nullptr, but redoPtr may be pointing to an edit log.



Undo pointer and Redo pointer

- Or, when no more redo-ing is possible, redoPtr is nullptr, but undoPtr may be pointing to an edit log.



Grouping editing operations

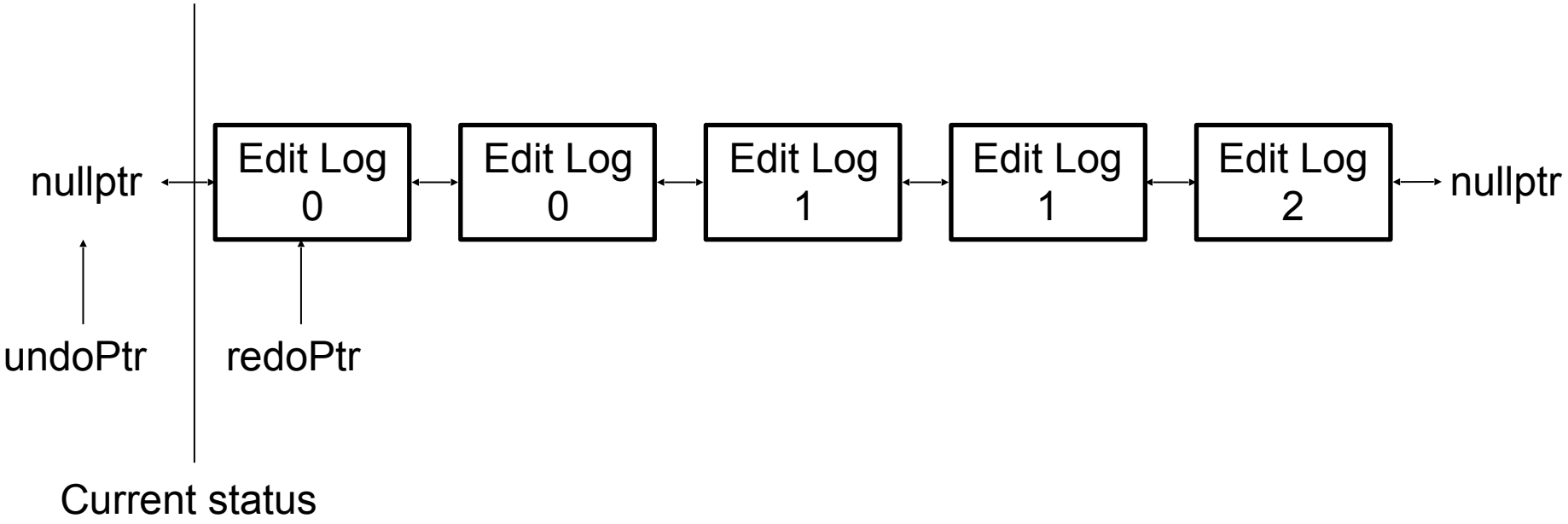
- Also, a sequence of editing operations may be a single operation from the user’s point of view.
- The operations must be grouped.

- How can you group a sequence of editing operations?

Grouping editing operations

- Undo counter
- Each editing operation is assigned a number.
- When an editing operation is applied, the number from a seed counter is assigned to the operation, and the seed is incremented (not grouped) or, stays the same (grouped.)
- If the operation should be grouped is controlled by a member variable of YsShellExtEdit class called incUndo.
- Variable incUndo can be 0 or 1.
- After each operation, incUndo is added to the seed.

Grouping editing operations



Grouping editing operations

- To group a sequence of operations,

```
auto incUndo=shl.PushStopIncUndo();
{
    Do a sequence of operations.
}
shl.PopIncUndo(incUndo);
```

- YsShellExtEdit::PushStopIncUndo() returns the current value of incUndo, and set it to zero.
- By doing it, you can also make nested grouped operations.

- But, there is a better way.

Grouping editing operations

- Use constructor and destructor.
- Instead of storing the value of incUndo, you can also do:

```
{
    YsShellExtEdit::StopIncUndo undoGuard(shl);
    /* Apply a sequence of operations. */
}
```

- I hope you can guess what's going on inside the constructor and destructor of StopIncUndo class.
- By doing so, any operations within the curly bracket are grouped.

- Let’s implement edge-collapsing operator, and also undo- and redo-ing.
- Change:
  #include <ysshellext.h>
  to
  #include <ysshellextedit.h>
- Change:
  YsShellExt to YsShellExtEdit

Reading from file must be a single modification. Needs to be done internally.

```
void FsGui3DMainCanvas::LoadModel(const char fn[])
{
    YsString str(fn);
    auto ext=str.GetExtension();
    if(0==ext.STRCMP(".SRF"))
    {
        YsFileIO::File fp(fn,"r");
        if(nullptr!=fp)
        {
            auto inStream=fp.InStream();
            shl.LoadSrf(inStream);
        }
    }
    else if(0==ext.STRCMP(".OBJ"))
    {
        YsFileIO::File fp(fn,"r");
        if(nullptr!=fp)
        {
            auto inStream=fp.InStream();
            shl.LoadObj(inStream);
        }
    }
    else if(0==ext.STRCMP(".STL"))
    {
        shl.LoadStl(fn);
        for(auto plHd : shl.AllPolygon())
        {
            shl.SetPolygonColor(plHd,YsBlue());
        }
    }
    else if(0==ext.STRCMP(".OFF"))
    {
        YsFileIO::File fp(fn,"r");
        if(nullptr!=fp)
        {
            auto inStream=fp.InStream();
            shl.LoadOff(inStream);
        }
    }

    RemakeVertexAttribArray();
    shl.EnableSearch();
    selectedVertex.clear();
    selVtxBuffer.clear();
}
```

Edge Collapse

- One of the topological transformation for a mesh.
- Easy to implement and fast.
- Commonly used for reducing the element count.

Edge Collapse

- Implementation for a triangular mesh:

  Edge Collapse vertex a⇒b

  1. Delete triangles sharing edge ab.
  2. For each polygon sharing vertex a, replace vertex a with b.

Edge Collapse

- Example: Collapse selected vertex [0] to [1].s

```
void FsGui3DMainCanvas::Edit_CollapseEdge(FsGuiPopUpMenuItem *)
{
    if(2==selectedVertex.size())
    {
        // Triangles using the edge must be deleted.
        for(auto plHd : shl.FindPolygonFromEdgePiece(selectedVertex[0],selectedVertex[1]))
        {
            shl.DeletePolygon(plHd);
        }
        // Reconnect selectedVertex[0] to selectedVertex[1]
        for(auto plHd : shl.FindPolygonFromVertex(selectedVertex[0]))
        {
            auto plVtHd=shl.GetPolygonVertex(plHd);
            for(auto &vtHd : plVtHd)
            {
                if(selectedVertex[0]==vtHd)
                {
                    vtHd=selectedVertex[1];
                }
            }
            shl.SetPolygonVertex(plHd,plVtHd);
        }
        RemakeVertexAttribArray();
        SetNeedRedraw(YSTRUE);
    }
    else
    {
        auto dlg=FsGuiDialog::CreateSelfDestructiveDialog<FsGuiMessageBoxDialog>();
        dlg->Make(L"Error",L"Select 2 vertices and try again.",L"YSOK",nullptr);
        AttachModalDialog(dlg);
    }
}
```

Adding Undo and Redo

- Add Edit->Undo and Edit->Redo

```
void FsGui3DMainCanvas::Edit_Undo(FsGuiPopUpMenuItem *)
{
    shl.Undo();
    RemakeVertexAttribArray();
}
void FsGui3DMainCanvas::Edit_Redo(FsGuiPopUpMenuItem *)
{
    shl.Redo();
    RemakeVertexAttribArray();
}
```

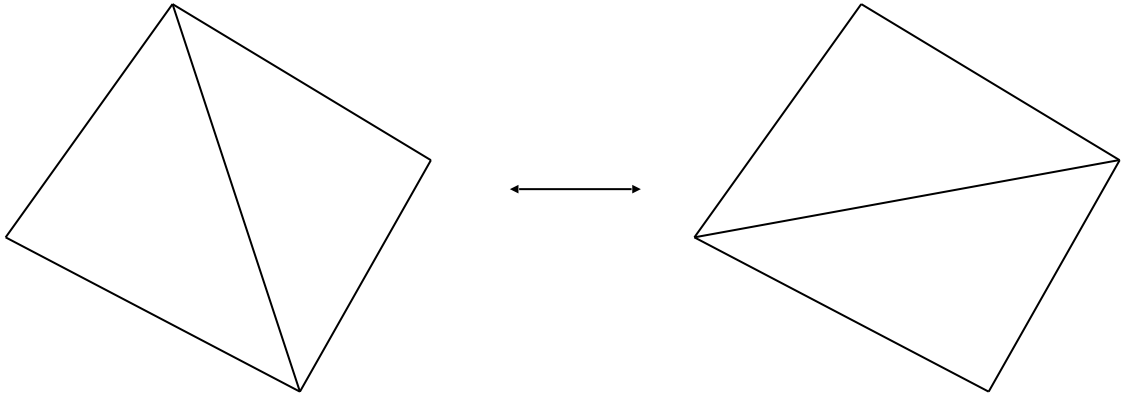
Adding Undo/Redo

- Then group editing operations:

```
YsShellExtEdit::StopIncUndo undoGuard(shl);
```

Edge-Swapping

- Edge-Collapsing and Edge-Swapping are the two most common topological transformations for a triangular mesh.
- Can be applied to two triangles sharing an edge.
- Can be used for improving the mesh quality.



Edge-Swapping

Implementation

1. Merge two triangles into a quadrilateral.
2. Split the quadrilateral into two with the new diagonal.

```
void FsGui3DMainCanvas::Edit_SwapEdge(FsGuiPopUpMenuItem *)
{
    YsShellExtEdit::StopIncUndo undoGuard(shl);
    if(2==selectedVertex.size())
    {
        YsShell::VertexHandle edVtHd[2]={selectedVertex[0],selectedVertex[1]};

        auto edPlHd=shl.FindPolygonFromEdgePiece(edVtHd[0],edVtHd[1]);
        if(2==edPlHd.GetN() &&
           3==shl.GetPolygonNumVertex(edPlHd[0]) &&
           3==shl.GetPolygonNumVertex(edPlHd[1]))
        {
            auto plVtHd0=shl.GetPolygonVertex(edPlHd[0]);
            auto plVtHd1=shl.GetPolygonVertex(edPlHd[1]);
            // First merge two polygons into a quadrilateral.
            int edIdxInPlg0=-1;
            for(int i=0; i<3; ++i)
            {
                if((plVtHd0[i]==edVtHd[0] && plVtHd0.GetCyclic(i+1)==edVtHd[1]) ||
                   (plVtHd0[i]==edVtHd[1] && plVtHd0.GetCyclic(i+1)==edVtHd[0]))
                {
                    edIdxInPlg0=i;
                    break;
                }
            }
            // Find which vertex should be merged.
            YsShell::VertexHandle vtHdToMerge=nullptr;
            for(int i=0; i<3; ++i)
            {
                if(plVtHd1[i]!=edVtHd[0] && plVtHd1[i]!=edVtHd[1])
                {
                    vtHdToMerge=plVtHd1[i];
                    break;
                }
            }
```

```
        if(nullptr!=vtHdToMerge && 0<=edIdxInPlg0)
        {
            plVtHd0.Insert(edIdxInPlg0+1,vtHdToMerge);
        }

        YsShell::VertexHandle triVtHd[2][3]=
        {
            {
                plVtHd0.GetCyclic(edIdxInPlg0+1),
                plVtHd0.GetCyclic(edIdxInPlg0+2),
                plVtHd0.GetCyclic(edIdxInPlg0+3)
            },
            {
                plVtHd0.GetCyclic(edIdxInPlg0+3),
                plVtHd0.GetCyclic(edIdxInPlg0+0),
                plVtHd0.GetCyclic(edIdxInPlg0+1)
            }
        };

        shl.SetPolygonVertex(edPlHd[0],3,triVtHd[0]);
        shl.SetPolygonVertex(edPlHd[1],3,triVtHd[1]);

        RemakeVertexAttribArray();
        SetNeedRedraw(YSTRUE);
    }
  }
}
```