Prototype of a 3D CAD system

Catia, Solidwork, ProE, UG, Inventor… lots of 3D CAD software are widely used in today's mechanical design area. They make the design process fairly different from that in decades ago, and they free the mechanical engineers from the tedious manual work.

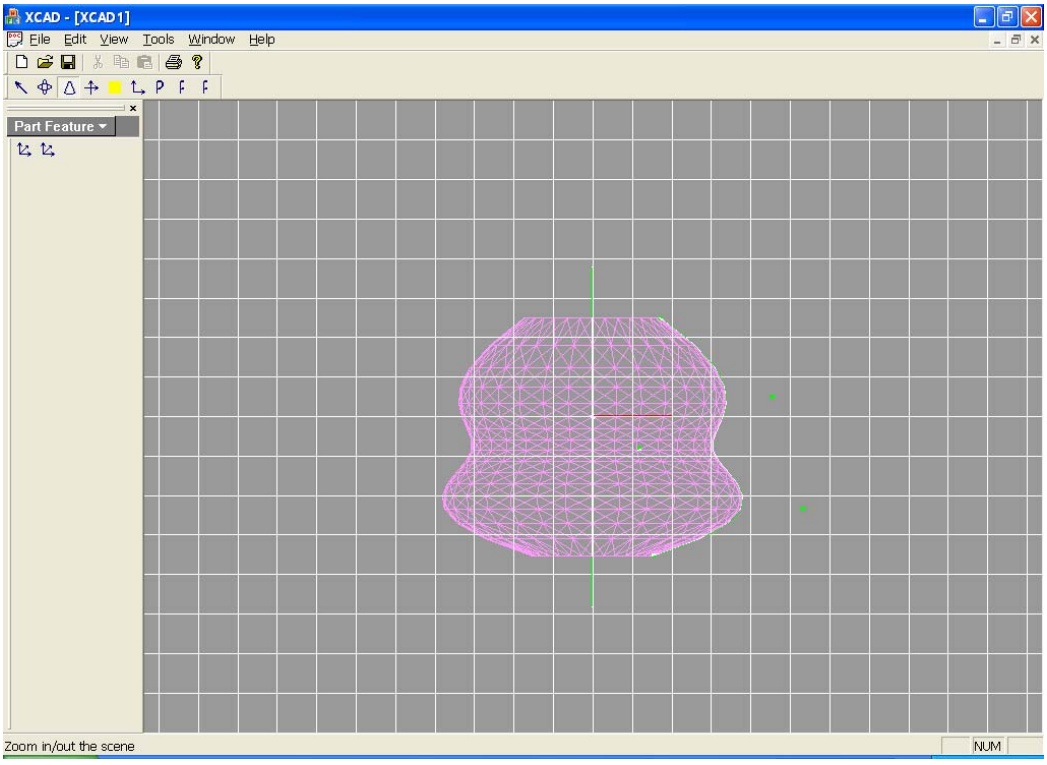
In order to touch every aspect of a CAD system, I am developing a prototype, XCAD, a mini-CAD application.

Unlike those widely used software, whose kernel is Parasolid, ACIS or some other mature library, I try to implement my own mathematics library and BREP library, which is simple but contains those usually used functions.

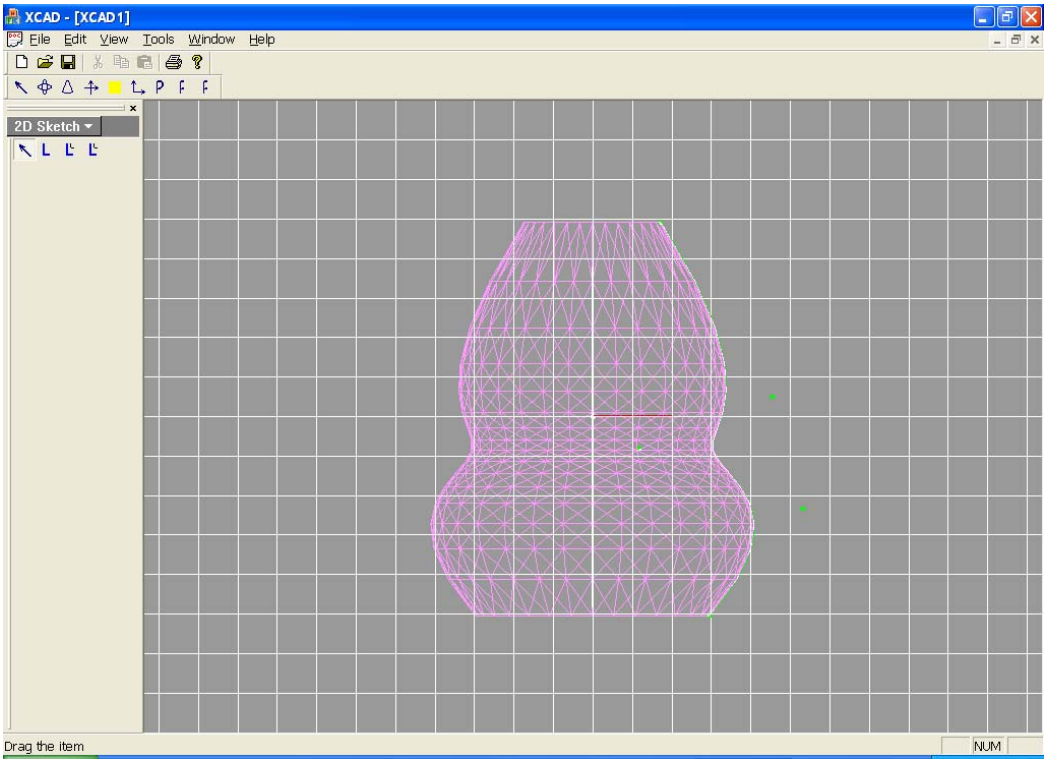
Another major character of XCAD is its intelligence, which is also called "driven by constraint" in many CAD systems. A sample of intelligence is given below:

A Sample

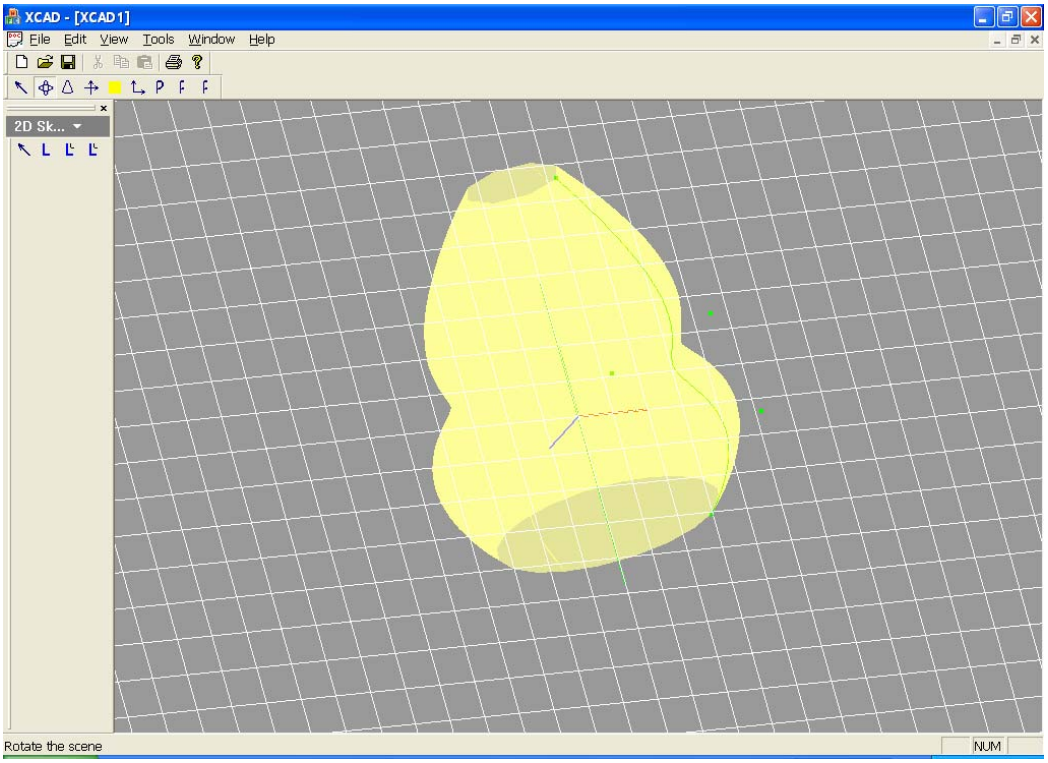
A vase's mesh. Revolve a NURBS curve around an axis.



After adjusting the profile of the vase (dragging the control points of the NURBS curve)



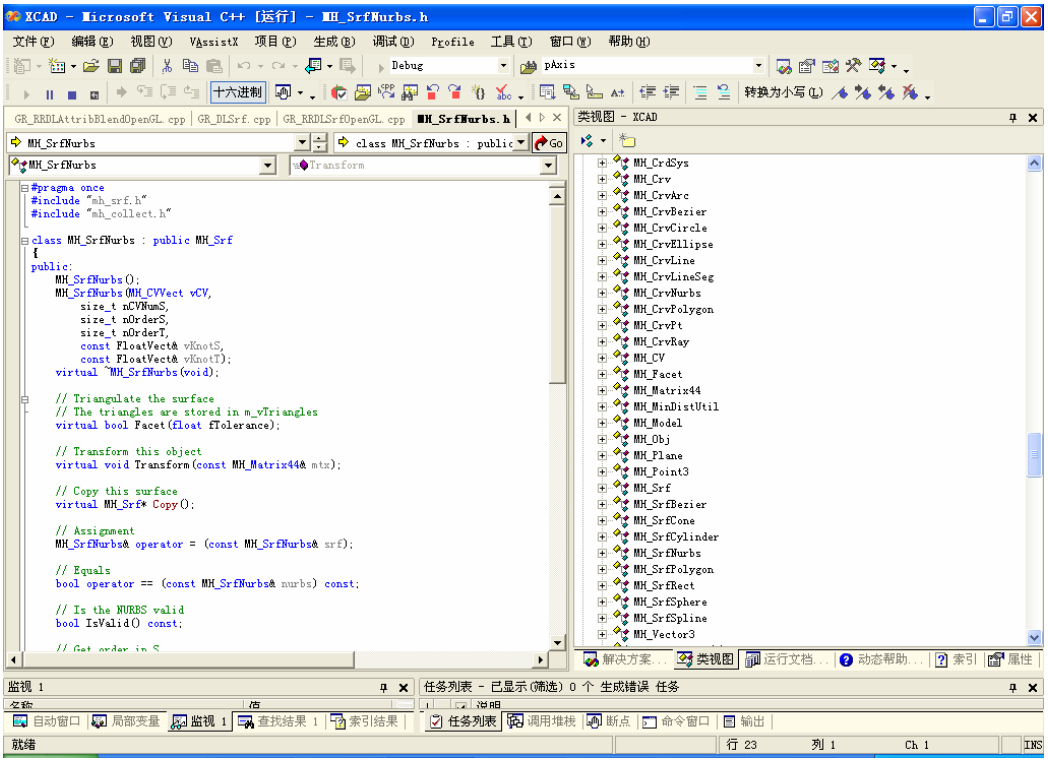
Rendering result (simple solution, just use OpenGL)



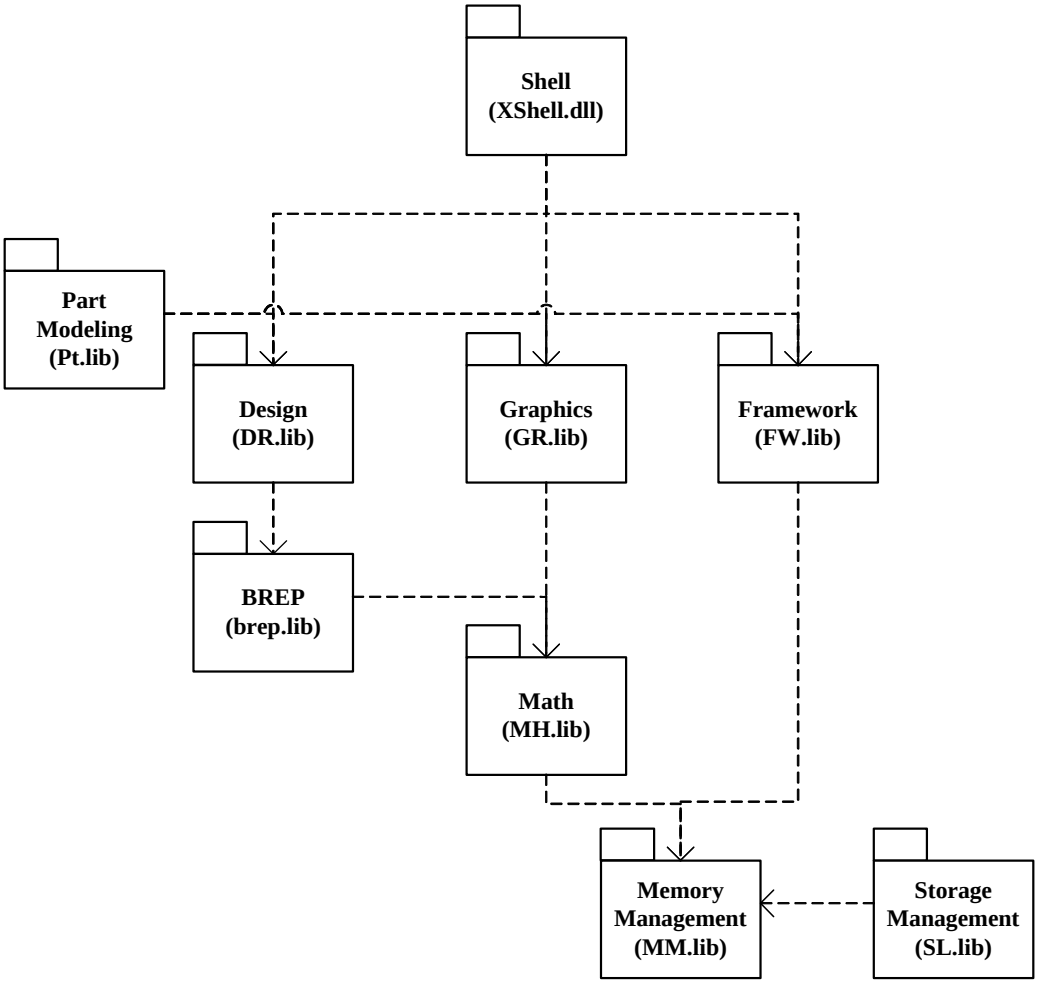
Development Environment & Performance Testing Tools

Windows XP & Visual Studio 2003 & AQ Time

Snapshot:


XCAD - Microsoft Visual C++ [运行] - MH_SrfNurbs.h
文件(F) 编辑(E) 视图(V) VAssistX 项目(P) 生成(B) 调试(D) Profile 工具(T) 窗口(W) 帮助(H)
Debug
pAxis
十六进制
转换为小写(L)
GR_RRDLAttribBlendOpenGL.cpp
GR_DLSrf.cpp
GR_RRDLSrfOpenGL.cpp
MH_SrfNurbs.h
类视图 - XCAD
MH_SrfNurbs
class MH_SrfNurbs : public
Go
Transform
#pragma once
#include "mh_srf.h"
#include "mh_collect.h"
class MH_SrfNurbs : public MH_Srf
{
public:
MH_SrfNurbs();
MH_SrfNurbs(MH_CVVect vCV,
size_t nCVNumS,
size_t nOrderS,
size_t nOrderT,
const FloatVect& vKnotS,
const FloatVect& vKnotT);
virtual ~MH_SrfNurbs(void);
// Triangulate the surface
// The triangles are stored in m_vTriangles
virtual bool Facet(float fTolerance);
// Transform this object
virtual void Transform(const MH_Matrix44& mtx);
// Copy this surface
virtual MH_Srf* Copy();
// Assignment
MH_SrfNurbs& operator = (const MH_SrfNurbs& srf);
// Equals
bool operator == (const MH_SrfNurbs& nurbs) const;
// Is the NURBS valid
bool IsValid() const;
MH_CrdSys
MH_Crv
MH_CrvArc
MH_CrvBezier
MH_CrvCircle
MH_CrvEllipse
MH_CrvLine
MH_CrvLineSeg
MH_CrvNurbs
MH_CrvPolygon
MH_CrvPt
MH_CrvRay
MH_CV
MH_Facet
MH_Matrix44
MH_MinDistUtil
MH_Model
MH_Obj
MH_Plane
MH_Point3
MH_Srf
MH_SrfBezier
MH_SrfCone
MH_SrfCylinder
MH_SrfNurbs
MH_SrfPolygon
MH_SrfRect
MH_SrfSphere
MH_SrfSpline
MH_Vector3
解决方案...
类视图
运行文档...
动态帮助...
索引
属性
监视 1
任务列表 - 已显示(筛选) 0 个 生成错误 任务
自动窗口
局部变量
监视 1
查找结果 1
索引结果
任务列表
调用堆栈
断点
命令窗口
输出
就绪
行 23
列 1
Ch 1
INS

Overview of components



Memory Management library (MM.lib)

The memory management library is the base for all the other components in the system. It determines the architecture of the system and most of the data structures very much. The main reason why to write this library is for handling the transaction issue, which is also usually called Undo/Redo issue.
The details of this solution will be introduced in the following sections.

Transaction

Let's think over the following question:
Initial State: Pointer p points to character 'a' (*p == 'a')
Operation: *p = 'b'; // Assign 'b' to *p
Undo: Undo the above operation
Requirement: We wants *p=='a' after the Undo operation. It means the character in *p is rolled back.

How to meet the requirement?

A Typical Solution: *transaction in object level*

Usually we implement transaction (undo/redo) by implementing a corresponding Undo/Redo action for each command.

The drawback of this solution is that it is hard to extend it to other applications and hard to maintain.

My Solution: *transaction in memory level*

If operation *p = 'b' can notify my application to backup that byte pointed by pointer p, then we have a chance to roll back that memory by a memory copy operation.

Then how can operation "*p = 'b'" trigger a notification? Fortunately, some Windows APIs give us that chance.

That series of APIs are:

LPVOID ***VirtualAlloc*** (LPVOID *lpAddress*, DWORD *dwSize*, DWORD *flAllocationType*, DWORD *flProtect*)

BOOL ***VirtualProtect*** (LPVOID *lpAddress*, DWORD *dwSize*, DWORD *flNewProtect*, PDWORD *lpflOldProtect*);

VirtualAlloc: This function reserves or commits a region of pages in the virtual address space of the calling process.

If parameter *flProtect* is PAGE_READONLY, then it enables read access to the committed region of pages. An attempt to write to the committed region results in an access violation. If the system differentiates between read-only accesses and executes access, an attempt to execute code in the committed region results in an access violation.

VirtualProtect: This function changes the access protection on a region of committed pages in the virtual address space of the calling process.

If parameter *flNewProtect* is PAGE_READONLY, then it enables read access to the committed region of pages. An attempt to write to the committed region results in an access violation. If the system differentiates between read-only accesses and executes access, an attempt to execute code in the committed region results in an access violation.

The ***access violation*** is the most important character we use in this solution.

Then the operation *p = 'b' can be convert to the following operations:

1. Lock the memory page that own the byte pointed by p. "Lock" means calling ***VirtualProtect*** with *flNewProtect* = PAGE_READONLY
2. Run *p = 'b'. Since that memory is read-only, this writing operation will cause an ***access violation*** exception. Once the application catches that exception, it will copy that page and then unlock that locked page. "Unlock" means calling ***VirtualProtect*** with *flNewProtect* = PAGE_READWRITE
3. Once Undo command is executed, we copy the backup memory back to previous page.

After step 3, we have made that memory rolled back.

A sample showing how to handle the access violation:

int ExceptionFilter (LPEXCEPTION_POINTERS e)

```
{
    // We are only interested in access violations (memory faults)
    if (e->ExceptionRecord->ExceptionCode != EXCEPTION_ACCESS_VIOLATION)
        return EXCEPTION_CONTINUE_SEARCH;

    // The exception information includes the type of access (read or write),
    // and the address of the fault
    bool writing = (e->ExceptionRecord->ExceptionInformation[0] != 0);
    void* addr = (void*) e->ExceptionRecord->ExceptionInformation [1];
    // TODO: Backup the page that contains addr and unlock the page
    return EXCEPTION_CONTINUE_EXECUTION;
}

int main ()
{
    int retval = 0;
    __try {
            retval = DoSomeWork ();
    } __except(ExceptionFilter(GetExceptionInformation())) {
            printf ("access violations \n");
    };
    return retval;
}
```

Storage Management library (SL.lib)

Usually we invoke CreateFile and WriteFile to create a file to save what we want to save. And later we invoke OpenFile and ReadFile to open a file to get what we want to get.
But in a large mechanical design, we usually face very large part files, and if we just want to read piece of information, such as a face's geometry, then if we use above APIs, megabytes should be read first and only bytes of information are really needed.
Obviously, above traditional solution is not suitable for a large CAD system. So I try structured storage technique in this prototype.

Structured Storage

Some introduction about this technique coming from MSDN:
Structured Storage provides file and data persistence in COM by treating a single file as a structured collection of objects known as storages and streams.
The purpose of Structured Storage is to reduce the performance penalties and overhead associated with storing separate objects in a single file. Structured Storage provides a solution by defining how to treat a single file entity as a structured collection of two types of objects—storages and streams—through a standard implementation called Compound Files. This lets the user interact with and manage a compound file as if it were a single file rather than a nested hierarchy of separate objects.

In this prototype, the file consists of the following streams:
Geometry Stream, which stores the geometry information.
Graphics Stream, which stores the graphics information.
Design Stream, which stores the design information.
The design of the system makes it easy to extend to consume other streams if needed.
Then in above example, we only need to open the geometry stream to read what we need.

Math library (MH.lib)

The Math library is the base of BREP, Design, and Graphics library.
It implements the main functions for NURBS curve and NURBS surface, which are the base of surface modeling.

Some basic classes are introduced below:

MH_Point3

Class for implementing basic functions of point
The class diagram is:

MH_Point3
+m_f[3] : float
+Transform(inout mtx : const MH_Matrix44) +MH_Point3() +MH_Point3(inout pt : const MH_Point3) +MH_Point3(inout A : const MH_Vector3) +MH_Point3(in x : float, in y : float, in z : float) +~MH_Point3() +Set(in x : float, in y : float, in z : float) +operator ==(inout pt : const MH_Point3) : bool +operator !=(inout pt : const MH_Point3) : bool +operator =(inout pt : const MH_Point3) : MH_Point3 & +operator +(inout A : const MH_Vector3) : MH_Point3 +operator -() : MH_Point3 +operator -(inout pt : const MH_Point3) : MH_Vector3 +operator *(in s : const float) : MH_Point3 +operator /(in s : const float) : MH_Point3 +operator +(inout A : const MH_Point3) : MH_Point3 +DistanceTo2(inout pt : const MH_Point3) : float +DistanceTo2(inout line : const MH_CrvLine, inout ptCross : MH_Point3) : float +DistanceTo2(inout lineSeg : const MH_CrvLineSeg, inout ptCross : MH_Point3) : float +DistanceTo2(inout plane : const MH_Plane, inout ptCross : MH_Point3) : float +DistanceTo2(inout fct : const MH_Facet, inout ptCross : MH_Point3) : float +Rotate(inout pt : const MH_Point3, inout vAxis : const MH_Vector3, in fAngle : float) : MH_Point3

MH_Vector3

Class for implementing basic functions of vector
The class diagram is:

MH_Vector3
+m_f[3] : float
+Transform(inout mtx : const MH_Matrix44) +MH_Vector3() +MH_Vector3(in x : float, in y : float, in z : float) +~MH_Vector3() +Set(in x : float, in y : float, in z : float) +Normalize() +IsValid() : bool +operator ==(inout A : const MH_Vector3) : bool +operator =(inout A : const MH_Vector3) : MH_Vector3 & +operator +(inout A : const MH_Vector3) : MH_Vector3 +operator -(inout A : const MH_Vector3) : MH_Vector3 +operator *(inout A : const MH_Vector3) : float +operator ^(inout A : const MH_Vector3) : MH_Vector3 +operator *(in s : const float) : MH_Vector3 +operator /(in s : const float) : MH_Vector3 +operator /(inout A : const MH_Vector3) : float +operator +=(in A : const MH_Vector3) +operator -=(in A : const MH_Vector3) +operator *=(in s : const float) +operator /=(in s : const float) +operator -() : MH_Vector3

MH_Plane

Class for implementing basic functions of plane

The class diagram is:

MH_Plane
-m_pt : MH_Point3 -m_vNormal : MH_Vector3
+Transform(inout mtx : const MH_Matrix44) +MH_Plane(inout pt : const MH_Point3, inout vNormal : const MH_Vector3) +MH_Plane(inout fct : const MH_Facet) +~MH_Plane() +GetPoint() : const MH_Point3 & +GetNormal() : const MH_Vector3 &

MH_Matrix

Class for implementing basic functions of 4*4 matrix

The class diagram is:

MH_Matrix44
-m_f[4][4] : float
+MH_Matrix44() +~MH_Matrix44() +Transform(inout mtx : const MH_Matrix44) +Identity() +operator *(inout m : const MH_Matrix44) : MH_Matrix44 +operator =(inout m : const MH_Matrix44) : MH_Matrix44 & +operator [](inout i : const int) : float & +operator [](inout i : const int) : const float & +Reverse() : MH_Matrix44 +Transpose() : MH_Matrix44 +Translate(inout v : const MH_Vector3) : MH_Matrix44

MH_CrvLine

Class for implementing basic functions of line

The class diagram is:

MH_CrvLine
-m_ptInLine : MH_Point3 -m_vDir : MH_Vector3
+Copy() : MH_Crv * +MH_CrvLine() +MH_CrvLine(inout ptInLine : const MH_Point3, inout vDir : const MH_Vector3) +MH_CrvLine(inout ray : const MH_CrvRay) +MH_CrvLine(inout lineSeg : const MH_CrvLineSeg) +~MH_CrvLine() +Transform(inout mtx : const MH_Matrix44) +operator =(inout crv : const MH_CrvLine) : MH_CrvLine & +GetPtInLine() : const MH_Point3 & +GetDir() : const MH_Vector3 & +CrossPoint(inout plane : const MH_Plane, in pPt : MH_Point3* = 0, in pU : float* = 0) : bool +DistanceTo2(inout line : const MH_CrvLine, inout ptCrossThis : MH_Point3, inout ptCrossThat : MH_Point3) : float +DistanceTo2(inout lineSeg : const MH_CrvLineSeg, inout ptCrossThis : MH_Point3, inout ptCrossThat : MH_Point3) : float +DistanceTo2(inout fct : const MH_Facet, inout ptCrossThis : MH_Point3, inout ptCrossThat : MH_Point3) : float +IsIn(inout pt : const MH_Point3) : bool +IsIn(inout line : const MH_CrvLine) : bool +IsIn(inout lineSeg : const MH_CrvLineSeg) : bool

MH_CrvLineSeg

Class for implementing basic functions of line segment

The class diagram is:

MH_CrvLineSeg
-m_ptFrom : MH_Point3 -m_ptTo : MH_Point3
+Tessellate(in fTolerance : float) : bool +Copy() : MH_Crv * +MH_CrvLineSeg() +MH_CrvLineSeg(inout ptFrom : const MH_Point3, inout ptTo : const MH_Point3) +MH_CrvLineSeg(inout ptFrom : const MH_Point3, inout vFrom2To : const MH_Vector3) +~MH_CrvLineSeg() +Transform(inout mtx : const MH_Matrix44) +HitTest(inout ray : const MH_CrvRay, inout ptHit : MH_Point3, in fTolerance : float) : bool +operator =(inout crv : const MH_CrvLineSeg) : MH_CrvLineSeg & +SetFrom(inout pt : const MH_Point3) +SetTo(inout pt : const MH_Point3) +GetFrom() : const MH_Point3 & +GetTo() : const MH_Point3 & +CheckPoint(inout pt : const MH_Point3, in piRegion : int* = 0, in pS : float* = 0) : bool +DistanceTo2(inout lineSeg : const MH_CrvLineSeg, inout ptCrossThis : MH_Point3, inout ptCrossThat : MH_Point3) : float +DistanceTo2(inout fct : const MH_Facet, inout ptCrossThis : MH_Point3, inout ptCrossThat : MH_Point3) : float

MH_CrvRay

Class for implementing basic functions of ray

The class diagram is:

MH_CrvRay
-m_ptFrom : MH_Point3 -m_vDir : MH_Vector3
+Copy() : MH_Crv * +MH_CrvRay() +MH_CrvRay(inout ptFrom : const MH_Point3, inout vDir : const MH_Vector3) +~MH_CrvRay() +Transform(inout mtx : const MH_Matrix44) +operator =(inout crv : const MH_CrvRay) : MH_CrvRay & +Set(inout ptFrom : const MH_Point3, inout vDir : const MH_Vector3) +GetFrom() : const MH_Point3 & +GetDir() : const MH_Vector3 & +CrossPoint(inout plane : const MH_Plane, in pPt : MH_Point3* = 0, in pU : float* = 0) : bool

MH_CrvBezier

Class for implementing basic functions of Bezier curve

The class diagram is:

MH_CrvBezier
-m_vCV : MH_CVVect -m_fU[2] : float
+Tessellate(in fTolerance : float) : bool +Copy() : MH_Crv * +MH_CrvBezier() +MH_CrvBezier(inout vCV : const MH_CVVect) +~MH_CrvBezier() +Transform(inout mtx : const MH_Matrix44) +operator =(inout crv : const MH_CrvBezier) : MH_CrvBezier & +SetDomain(in u0 : float, in u1 : float) +GetDomain(inout u0 : float, inout u1 : float) +SetCVs(inout vCV : const MH_CVVect) +GetCVs() : const MH_CVVect & +GetPoint(in u : float, inout pt3 : MH_Point3) : bool +DegreeElevate(in n : size_t) : bool +Divide(in u : float, inout bezier1 : MH_CrvBezier, inout bezier2 : MH_CrvBezier) : bool +Subdivision(in fTol : float, inout vCrvPt : MH_CrvPtVect, in pvCrvBezier : MH_CrvBezierVect* = 0) : bool -DegreeElevate() : bool -Subdivision(in fTol : float, inout bezier : const MH_CrvBezier, inout vCrvPt : MH_CrvPtVect, in pvCrvBezier : MH_CrvBezierVect*) : bool

MH_CrvNurbs

Class for implementing basic functions of NURBS curve

These functions include:

GetPoint:	Evaluate point at parameter u
InsertKnot:	Insert a knot multi-times
RemoveKnot:	Remove a knot multi-times
ElevateDegree:	Elevate degree
Divide:	Divide a NURBS into two NURBS curves at parameter u
ConvertToBezier:	Convert the NURBS curves into Bezier curves
Subdivision:	Subdivide a NURBS curve into poly-line

In order to render a curve, we will tessellate the curve into poly-line first.

Function Subdivision will help to do it.

The class diagram is:

MH_CrvNurbs
-m_nOrder : size_t -m_vCV : MH_CVVect -m_vKnot : FloatVect
+Tessellate(in fTolerance : float) : bool +Copy() : MH_Crv * +Dump() +MH_CrvNurbs() +MH_CrvNurbs(inout nurbs : const MH_CrvNurbs) +MH_CrvNurbs(in nOrder : size_t, inout vCV : const MH_CVVect, inout vKnot : const FloatVect) +~MH_CrvNurbs() +Transform(inout mtx : const MH_Matrix44) +operator =(inout crv : const MH_CrvNurbs) : MH_CrvNurbs & +operator ==(inout nurbs : const MH_CrvNurbs) : bool +IsValid() : bool +SetInfo(in nOrder : size_t, inout vCV : const MH_CVVect, inout vKnot : const FloatVect) +GetInfo(inout nOrder : size_t, inout vCV : MH_CVVect, inout vKnot : FloatVect) +GetOrder() : size_t +GetCVs() : const MH_CVVect & +GetKnots() : const FloatVect & +SetCV(in nIndex : size_t, inout cv : const MH_CV) : bool +GetCV(in nIndex : size_t, inout cv : MH_CV) : bool +GetPoint(in u : float, inout cv : MH_CV) : bool +GetDrv(in k : size_t, in u : float) : MH_Point3Vect +InsertKnot(in u : float) : bool +RemoveKnot(in u : float) : bool +InsertKnot(in u : float, in h : int) : bool +ElevateDegree() : bool +Divide(in u : float, inout nurb1 : MH_CrvNurbs, inout nurb2 : MH_CrvNurbs) : bool +Subdivision(in fTol : float, inout vCrvPt : MH_CrvPtVect, in pvCrvNurbs : MH_CrvNurbsVect* = 0) : bool +ConvertToBeziers(inout beziers : MH_CrvBezierVect, inout nurbs : MH_CrvNurbs = MH_CrvNurbs()) : bool +Revolve(inout crvLine : const MH_CrvLine) : MH_SrfNurbs -Subdivision(in fTol : float, inout nurbs : const MH_CrvNurbs, inout vCrvPt : MH_CrvPtVect, in pvCrvNurbs : MH_CrvNurbsVect*) : bool

MH_SrfBezier

Class for implementing basic functions of Bezier surface

The class diagram is:

MH_SrfBezier
-m_vCV : MH_CVVect -m_nCVNumS : size_t
+Transform(inout mtx : const MH_Matrix44) +Facet(in fTolerance : float) : bool +Copy() : MH_Srf * +MH_SrfBezier() +~MH_SrfBezier() +operator =(inout srf : const MH_SrfBezier) : MH_SrfBezier &

MH_SrfNurbs

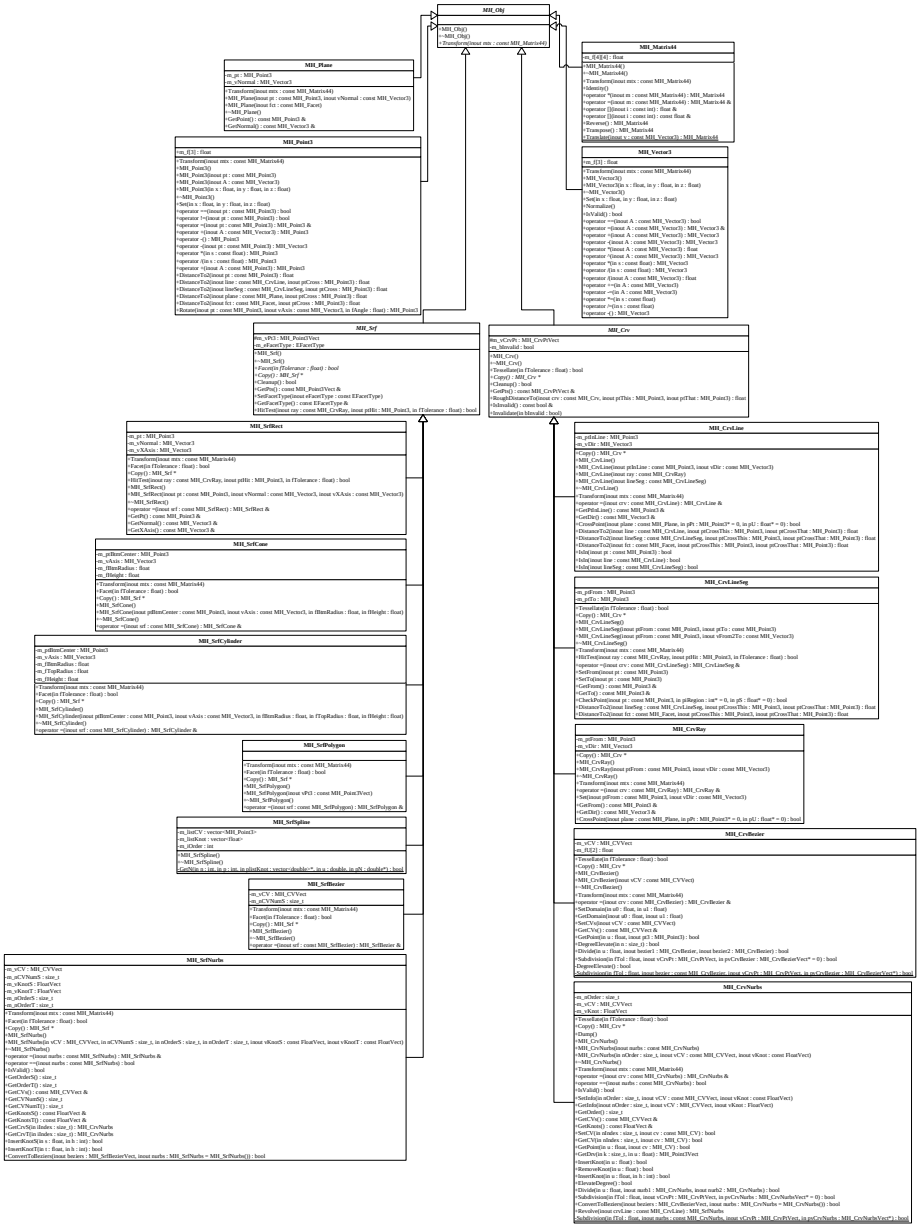
Class for implementing basic functions of NURBS surface

In order to render a surface, we will facet the surface into triangles first.

Function Facet will help to do it.

MH_SrfNurbs
-m_vCV : MH_CVVect -m_nCVNumS : size_t -m_vKnotS : FloatVect -m_vKnotT : FloatVect -m_nOrderS : size_t -m_nOrderT : size_t
+Transform(inout mtx : const MH_Matrix44) +Facet(in fTolerance : float) : bool +Copy() : MH_Srf * +MH_SrfNurbs() +MH_SrfNurbs(in vCV : MH_CVVect, in nCVNumS : size_t, in nOrderS : size_t, in nOrderT : size_t, inout vKnotS : const FloatVect, inout vKnotT : const FloatVect) +~MH_SrfNurbs() +operator =(inout nurbs : const MH_SrfNurbs) : MH_SrfNurbs & +operator ==(inout nurbs : const MH_SrfNurbs) : bool +IsValid() : bool +GetOrderS() : size_t +GetOrderT() : size_t +GetCVs() : const MH_CVVect & +GetCVNumS() : size_t +GetCVNumT() : size_t +GetKnotsS() : const FloatVect & +GetKnotsT() : const FloatVect & +GetCrvS(in iIndex : size_t) : MH_CrvNurbs +GetCrvT(in iIndex : size_t) : MH_CrvNurbs +InsertKnotS(in s : float, in h : int) : bool +InsertKnotT(in t : float, in h : int) : bool +ConvertToBeziers(inout beziers : MH_SrfBezierVect, inout nurbs : MH_SrfNurbs = MH_SrfNurbs()) : bool

The hierarchy of the main classes is given below:



BREP library (BREP.lib)

The BREP library is the base of Design library.

Some basic classes are introduced below:

BREP_Body

Class for implementing basic functions of body

The class diagram is:

BREP_Body
-m_mtxTransform : MH_Matrix44 -m_vLump : BREP_LumpPtrVect -m_vWire : BREP_WirePtrVect
+BREP_Body() +~BREP_Body() +Dump(in lpFileName : LPCTSTR, in bRunDotty : BOOL = 1) +Copy() : BREP_Body * +SetTransformation(inout mtx : const MH_Matrix44) +GetTransformation() : const MH_Matrix44 & +AddLump(inout pLump : const SL_Ptr<BREP_Lump>) +GetLumps() : const BREP_LumpPtrVect & +AddWire(inout pWire : const SL_Ptr<BREP_Wire>) +GetWires() : const BREP_WirePtrVect & +GetPoint() : MH_Point3

BREP_Lump

Class for implementing basic functions of lump

The class diagram is:

BREP_Lump
-m_pBody : SL_Ptr<BREP_Body> -m_vShell : BREP_ShellPtrVect
+BREP_Lump(inout pBody : const SL_Ptr<BREP_Body>) +BREP_Lump(inout pBody : const SL_Ptr<BREP_Body>, inout vShell : const BREP_ShellPtrVect) +~BREP_Lump() +Copy(inout pBody : const SL_Ptr<BREP_Body>) : BREP_Lump * +GetBody() : const SL_Ptr<BREP_Body> & +AddShell(inout pShell : const SL_Ptr<BREP_Shell>) +GetShells() : const BREP_ShellPtrVect &

BREP_Shell

Class for implementing basic functions of shell

The class diagram is:

BREP_Shell
-m_pLump : SL_Ptr<BREP_Lump> -m_vFace : BREP_FacePtrVect -m_vWire : BREP_WirePtrVect
+BREP_Shell(inout pLump : const SL_Ptr<BREP_Lump>) +BREP_Shell(inout pLump : const SL_Ptr<BREP_Lump>, inout vFace : const BREP_FacePtrVect, inout vWire : const BREP_WirePtrVect) +~BREP_Shell() +Copy(inout pLump : const SL_Ptr<BREP_Lump>) : BREP_Shell * +GetLump() : const SL_Ptr<BREP_Lump> & +AddFace(inout pFace : const SL_Ptr<BREP_Face>) +GetFaces() : const BREP_FacePtrVect & +AddWire(inout pWire : const SL_Ptr<BREP_Wire>) +GetWires() : const BREP_WirePtrVect &

BREP_Face

Class for implementing basic functions of face

The class diagram is:

BREP_Face
-m_pShell : SL_Ptr<BREP_Shell> -m_pSrf : SL_Ptr<MH_Srf> -m_vLoop : BREP_LoopPtrVect
+BREP_Face(inout pShell : const SL_Ptr<BREP_Shell>) +BREP_Face(inout pShell : const SL_Ptr<BREP_Shell>, inout pSrf : const SL_Ptr<MH_Srf>, inout vLoop : const BREP_LoopPtrVect) +~BREP_Face() +Copy(inout pShell : const SL_Ptr<BREP_Shell>) : BREP_Face * +GetShell() : const SL_Ptr<BREP_Shell> & +GetSrf() : const SL_Ptr<MH_Srf> & +AddLoop(inout pLoop : const SL_Ptr<BREP_Loop>) +GetLoops() : const BREP_LoopPtrVect &

BREP_Loop

Class for implementing basic functions of loop

The class diagram is:

BREP_Loop
-m_pFace : SL_Ptr<BREP_Face> -m_vCoEdge : BREP_CoEdgePtrVect
+BREP_Loop(inout pFace : const SL_Ptr<BREP_Face>) +BREP_Loop(inout pFace : const SL_Ptr<BREP_Face>, inout vCoEdge : const BREP_CoEdgePtrVect) +~BREP_Loop() +Copy(inout pFace : const SL_Ptr<BREP_Face>) : BREP_Loop * +GetFace() : const SL_Ptr<BREP_Face> & +AddCoEdge(inout pCoEdge : const SL_Ptr<BREP_CoEdge>) +GetCoEdges() : const BREP_CoEdgePtrVect &

BREP_CoEdge

Class for implementing basic functions of CoEdge

The class diagram is:

BREP_CoEdge
-m_pEntity : SL_Ptr<BREP_Entity> -m_pEdge : SL_Ptr<BREP_Edge> -m_bRev : bool
+BREP_CoEdge(inout pEntity : const SL_Ptr<BREP_Entity>) +BREP_CoEdge(inout pEntity : const SL_Ptr<BREP_Entity>, inout pEdge : const SL_Ptr<BREP_Edge>) +~BREP_CoEdge() +Copy(inout pEntity : const SL_Ptr<BREP_Entity>) : BREP_CoEdge * +GetEntity() : const SL_Ptr<BREP_Entity> & +SetEdge(inout pEdge : const SL_Ptr<BREP_Edge>) +GetEdge() : const SL_Ptr<BREP_Edge> & +SetRev(in bRev : bool) +GetRev() : bool

BREP_Edge

Class for implementing basic functions of edge

The class diagram is:

BREP_Edge
-m_pCoEdge : SL_Ptr<BREP_CoEdge> -m_pCrv : SL_Ptr<MH_Crv> -m_pVtxStart : SL_Ptr<BREP_Vertex> -m_pVtxEnd : SL_Ptr<BREP_Vertex>
+BREP_Edge(inout pCoEdge : const SL_Ptr<BREP_CoEdge>) +BREP_Edge(inout pCoEdge : const SL_Ptr<BREP_CoEdge>, inout pCrv : const SL_Ptr<MH_Crv>, inout pVtxStart : const SL_Ptr<BREP_Vertex>, inout pVtxEnd : const SL_Ptr<BREP_Vertex>) +~BREP_Edge() +Copy(inout pCoEdge : const SL_Ptr<BREP_CoEdge>) : BREP_Edge * +SetCoEdge(inout pCoEdge : const SL_Ptr<BREP_CoEdge>) +GetCoEdge() : const SL_Ptr<BREP_CoEdge> & +SetCrv(inout pCrv : const SL_Ptr<MH_Crv>) +GetCrv() : const SL_Ptr<MH_Crv> & +SetStartVertex(inout pVertex : const SL_Ptr<BREP_Vertex>) +GetStartVertex() : const SL_Ptr<BREP_Vertex> & +SetEndVertex(inout pVertex : const SL_Ptr<BREP_Vertex>) +GetEndVertex() : const SL_Ptr<BREP_Vertex> &

BREP_Vertex

Class for implementing basic functions of vertex

The class diagram is:

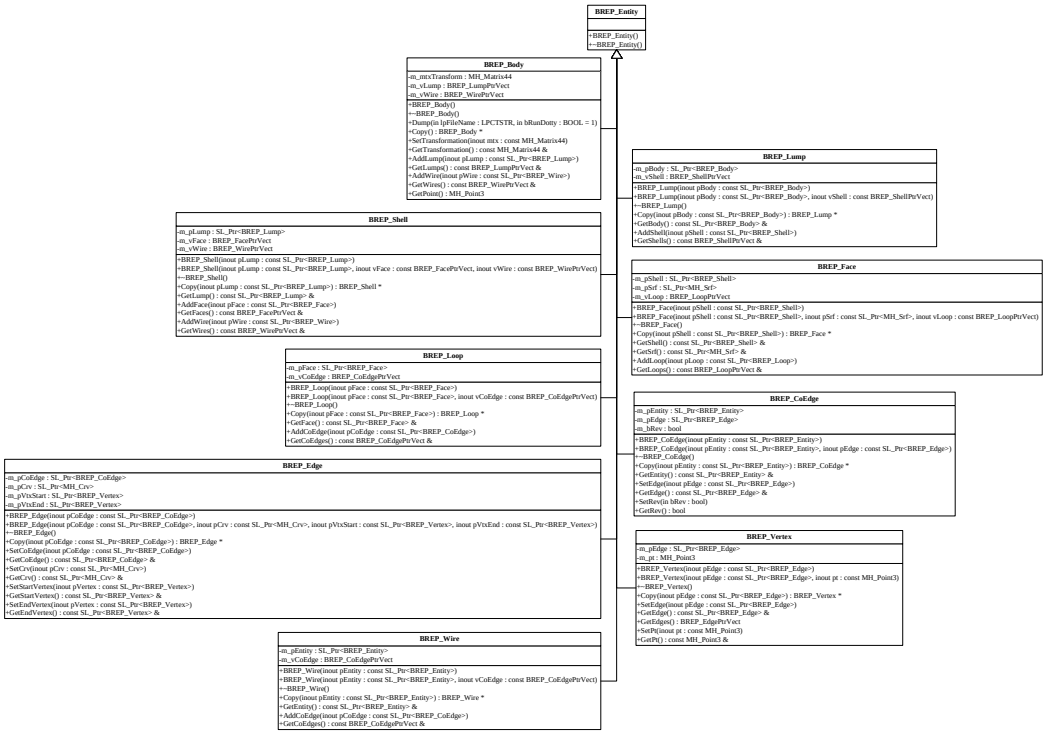
BREP_Vertex
-m_pEdge : SL_Ptr<BREP_Edge> -m_pt : MH_Point3
+BREP_Vertex(inout pEdge : const SL_Ptr<BREP_Edge>) +BREP_Vertex(inout pEdge : const SL_Ptr<BREP_Edge>, inout pt : const MH_Point3) +~BREP_Vertex() +Copy(inout pEdge : const SL_Ptr<BREP_Edge>) : BREP_Vertex * +SetEdge(inout pEdge : const SL_Ptr<BREP_Edge>) +GetEdge() : const SL_Ptr<BREP_Edge> & +GetEdges() : BREP_EdgePtrVect +SetPt(inout pt : const MH_Point3) +GetPt() : const MH_Point3 &

BREP_Wire

Class for implementing basic functions of wire

The class diagram is:

BREP_Wire
-m_pEntity : SL_Ptr<BREP_Entity> -m_vCoEdge : BREP_CoEdgePtrVect
+BREP_Wire(inout pEntity : const SL_Ptr<BREP_Entity>) +BREP_Wire(inout pEntity : const SL_Ptr<BREP_Entity>, inout vCoEdge : const BREP_CoEdgePtrVect) +~BREP_Wire() +Copy(inout pEntity : const SL_Ptr<BREP_Entity>) : BREP_Wire * +GetEntity() : const SL_Ptr<BREP_Entity> & +AddCoEdge(inout pCoEdge : const SL_Ptr<BREP_CoEdge>) +GetCoEdges() : const BREP_CoEdgePtrVect &

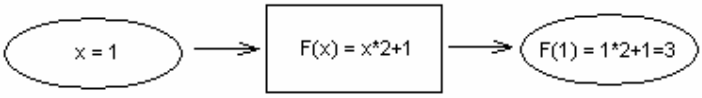


A utility, BREP_Util is developed to do some usually used operations (The Boolean operation is still in process), and the class diagram is given below:

BREP_Util
+DeleteEntity(in pEntity : BREP_Entity*) : HRESULT
+CreateVertex(in pEdge : BREP_Edge*, inout pt : const MH_Point3, inout pVertex : BREP_Vertex*) : HRESULT
+CreateEdge(in pCoEdge : BREP_CoEdge*, in pCrv : MH_Crv*, in pVtxStart : BREP_Vertex*, in pVtxEnd : BREP_Vertex*, inout pEdge : BREP_Edge*) : HRESULT
+CreateCoEdge(in pEntity : BREP_Entity*, in pEdge : BREP_Edge*, inout pCoEdge : BREP_CoEdge*) : HRESULT
+CreateLoop(in pFace : BREP_Face*, inout vCoEdge : const BREP_CoEdgePtrVect, inout pLoop : BREP_Loop*) : HRESULT
+CreateWire(in pEntity : BREP_Entity*, inout vCoEdge : const BREP_CoEdgePtrVect, inout pWire : BREP_Wire*) : HRESULT
+CreateFace(in pShell : BREP_Shell*, in pSrf : MH_Srf*, inout vLoop : const BREP_LoopPtrVect, inout pFace : BREP_Face*) : HRESULT
+CreateShell(in pLump : BREP_Lump*, inout vFace : const BREP_FacePtrVect, inout vWire : const BREP_WirePtrVect, inout pShell : BREP_Shell*) : HRESULT
+CreateLump(in pBody : BREP_Body*, inout vShell : const BREP_ShellPtrVect, inout pLump : BREP_Lump*) : HRESULT
+CreateBody(inout pBody : BREP_Body*) : HRESULT
+GetEdges(inout vCoEdge : const BREP_CoEdgePtrVect, inout vEdge : BREP_EdgePtrVect) : HRESULT
+GetEdges(in pLoop : BREP_Loop*, inout vEdge : BREP_EdgePtrVect) : HRESULT
+GetEdges(in pWire : BREP_Wire*, inout vEdge : BREP_EdgePtrVect) : HRESULT
+GetEdges(in pFace : BREP_Face*, inout vEdge : BREP_EdgePtrVect) : HRESULT
+GetEdges(in pShell : BREP_Shell*, inout vEdge : BREP_EdgePtrVect) : HRESULT
+GetEdges(in pLump : BREP_Lump*, inout vEdge : BREP_EdgePtrVect) : HRESULT
+GetEdges(in pBody : BREP_Body*, inout vEdge : BREP_EdgePtrVect) : HRESULT
+GetCoEdges(in pLoop : BREP_Loop*, inout vCoEdge : BREP_CoEdgePtrVect) : HRESULT
+GetCoEdges(in pWire : BREP_Wire*, inout vCoEdge : BREP_CoEdgePtrVect) : HRESULT
+GetCoEdges(in pFace : BREP_Face*, inout vCoEdge : BREP_CoEdgePtrVect) : HRESULT
+GetCoEdges(in pShell : BREP_Shell*, inout vCoEdge : BREP_CoEdgePtrVect) : HRESULT
+GetCoEdges(in pLump : BREP_Lump*, inout vCoEdge : BREP_CoEdgePtrVect) : HRESULT
+GetCoEdges(in pBody : BREP_Body*, inout vCoEdge : BREP_CoEdgePtrVect) : HRESULT
+GetFaces(in pShell : BREP_Shell*, inout vFace : BREP_FacePtrVect) : HRESULT
+GetFaces(in pLump : BREP_Lump*, inout vFace : BREP_FacePtrVect) : HRESULT
+GetFaces(in pBody : BREP_Body*, inout vFace : BREP_FacePtrVect) : HRESULT
+GetWires(in pShell : BREP_Shell*, inout vWire : BREP_WirePtrVect) : HRESULT
+GetWires(in pLump : BREP_Lump*, inout vWire : BREP_WirePtrVect) : HRESULT
+GetWires(in pBody : BREP_Body*, inout vWire : BREP_WirePtrVect) : HRESULT
+GetShells(in pLump : BREP_Lump*, inout vShell : BREP_ShellPtrVect) : HRESULT
+GetShells(in pBody : BREP_Body*, inout vShell : BREP_ShellPtrVect) : HRESULT
+GetLumps(in pBody : BREP_Body*, inout vLump : BREP_LumpPtrVect) : HRESULT
+BuildWire(inout vCrv : const MH_CrvPtrVect, inout vPt3 : const MH_Point3Vect, inout pWire : BREP_Wire*) : HRESULT
+BuildRevolve(in pWire : BREP_Wire*, in pAxis : BREP_Edge*, inout pBody : BREP_Body*) : HRESULT
+IsClosed(in pWire : BREP_Wire*, inout bClosed : bool) : HRESULT
+BuildRevolveFace(in pProfile : BREP_Edge*, in pAxis : BREP_Edge*, in pShell : BREP_Shell*, inout pFace : BREP_Face*) : HRESULT
+BuildRevolveFaceBody(in pProfile : BREP_Edge*, in pAxis : BREP_Edge*, inout pFaceBody : BREP_Body*) : HRESULT
+BuildRevolveBody(in pProfile : BREP_Entity*, in pAxis : BREP_Edge*, inout pBody : BREP_Body*) : HRESULT
+DeleteBody(in pBody : BREP_Body*) : HRESULT
+Boolean(in pTool : BREP_Body*, in pBlank : BREP_Body*, in eType : EBooleanType) : HRESULT
+BuildSweepBody(in pProfile : BREP_Entity*, in pPath : BREP_Entity*, inout pBody : BREP_Body*) : HRESULT
+BuildSweepFace(in pProfile : BREP_Edge*, in pPath : BREP_Edge*, inout pFace : BREP_Face*) : HRESULT
+BuildRevolveFace(in pProfile : BREP_Edge*, in pPath : BREP_Edge*, inout pFace : BREP_Face*) : HRESULT
+GetFaces(in pEntity : BREP_Entity*, inout pvFaces : BREP_FacePtrVect) : HRESULT
+Facet(in pFace : BREP_Face*, in fTol : float, inout pSrfPolygon : MH_SrfPolygon*) : HRESULT

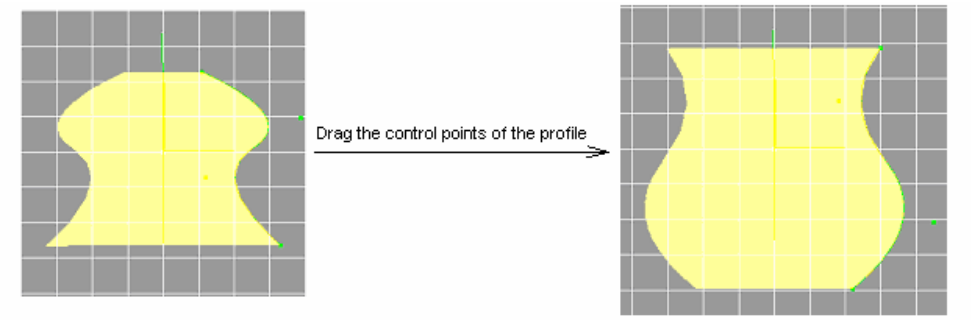
Design library for Intelligence (DR.lib)

In this sub-system, we define two important concepts: ***State*** and ***Constraint.***
Let's explain these two concepts through a quite abstract model of this Design Sub-System:

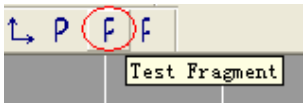


In the diagram given above, "x=1" is the input of "F(x) = x*2+1" whose output is "F(1)=1*2+1=3". Then we call the machine (F(x) = x*2+1) that has the ability to calculate as "***Constraint***", and other input(s) and output(s) of the constraint as "***State***".
The diagram is names as ***state-constraint-relationship diagram***.
Ellipse is treated as a state and rectangle is treated as a constraint.
Once an input of the constraint is changed, the system will force the constraint to compute to update its outputs – This auto-computation makes the system intelligent.

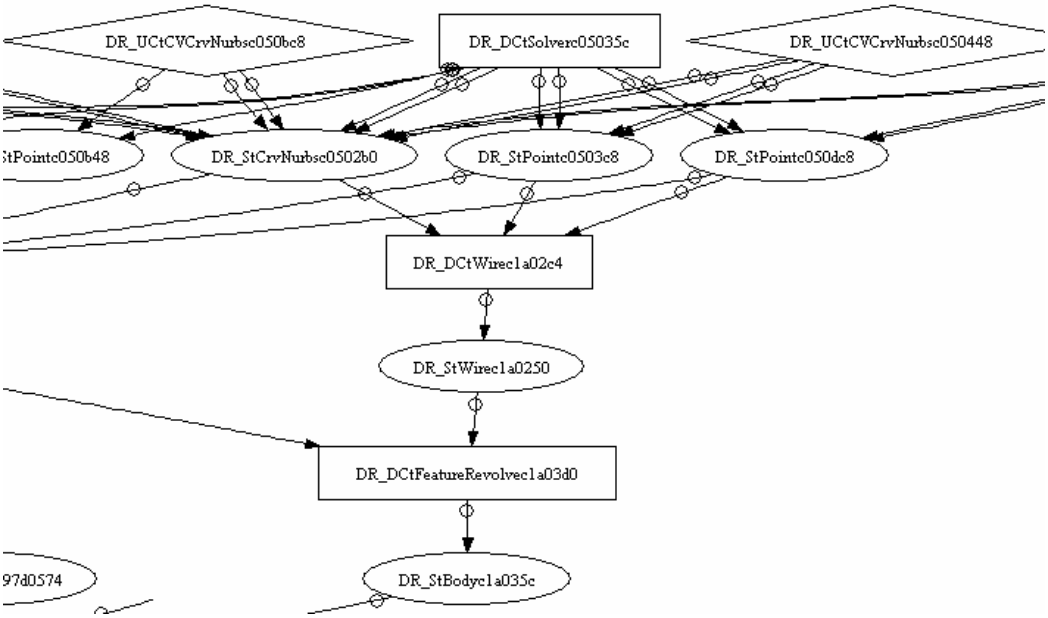
Then let's recall the example given in the 1st section of the document, a true example:



The surface feature (the vase) will be updated *automatically* once the shape of its profile curve is changed.
The state is the profile curve and the constraint is the machine that computes the new shape of the vase.

Use the command "Test fragment" 

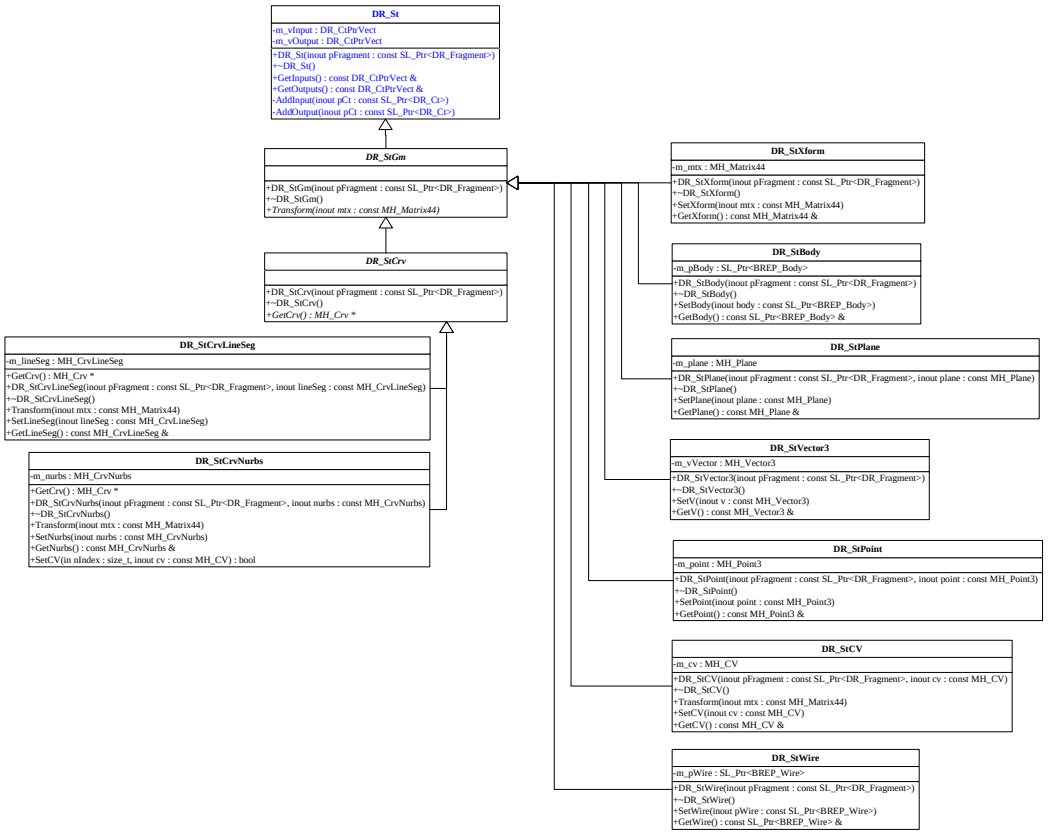
, we can get the state-constraint-relationship diagram as below (The number following the object is the address of that object):



DR_DctFeatureRevolve is the constraint used to compute the revolved surface.

State

Some typical states are listed below. It is easy to add new kinds of states into the system.



Constraint

In fact, there are two kinds of constraints:

Directed Constraint.

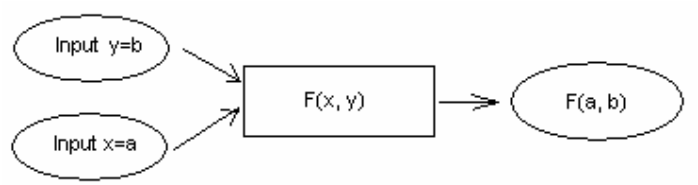
Undirected Constraint.

We will give the abstract models of these two kinds of constraints below.

So far, the solution to implement the undirected constraint is in process.

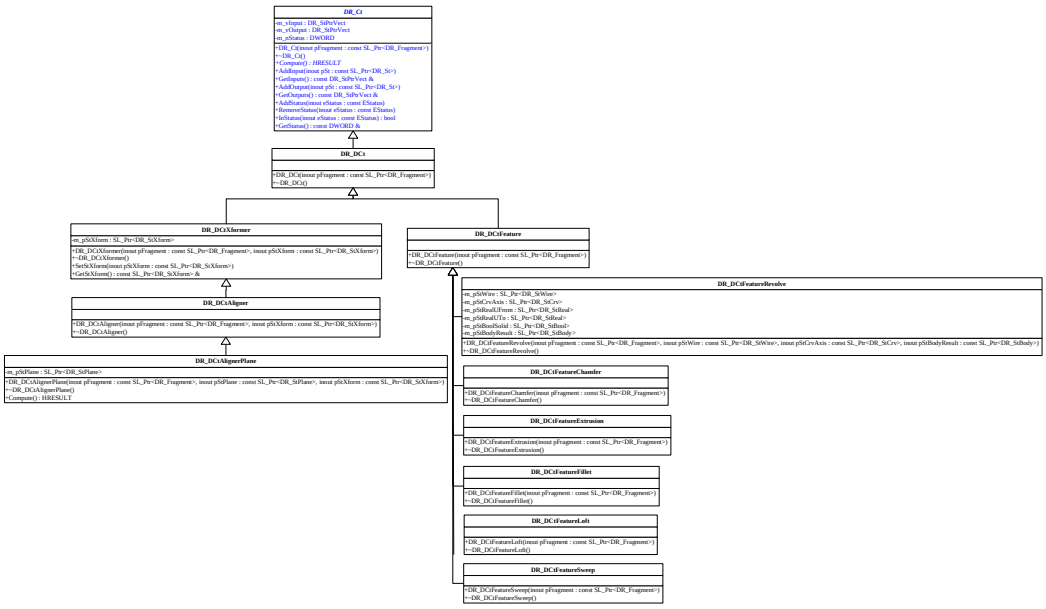
Directed Constraint

The abstract diagram for this kind of constraint is



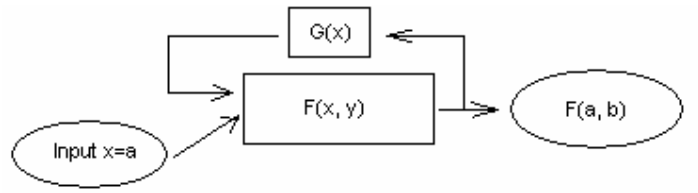
The main character is that there is no feedback to the constraint F(x,y).

Some typical directed constraints are listed below:



Undirected Constraint

Let's consider the following diagram:



The main character of this diagram is that there is feedback to the constraint F(x,y).

If the system is stable, then it means:

b = G(F(a,b))

In a word, the output of the constraint is also constrained by its output.

The main point to implement the undirected constraint is to solve nonlinear equations. Newton-Raphson method should be a mature solution.

Graphics library (GR.lib)

There are many libraries can help us to render objects in 3D model space, such as OpenGL and Direct3D.
In this application, we use OpenGL to render the scene. And the design can easily consume other graphics engines to render the scene.

The design is based on some valuable data structures:

GR_DL

Class used to stand for a display list. Each display list can contain some attributes, such as the color, blend, etc.
Specific class can be derived from this class to stand for a point, curve, surface, etc.
The class diagram is:

GR_DL
-m_pRRDL : GR_RRDL * -m_pParentNode : SL_Ptr<GR_DLNode> -m_vDLAttrib : GR_DLAttribPtrVect -m_mtxTransform : MH_Matrix44 -m_bInvalid : bool
+GR_DL(inout pParentNode : const SL_Ptr<GR_DLNode>, in bIsDL : bool = 1) +~GR_DL() +GetType() : EType +DECLARE_CREATOR(in 参数1 : GR_DL) : int +Save(inout stream : const SL_Stream) : HRESULT +Load(inout stream : const SL_Stream) : HRESULT +Detach() : bool +Invalidate(in bInvalid : bool = true) +Render() : bool +HitTest(inout ray : const MH_CrvRay, inout ptHit : MH_Point3, in fTolerance : float) : bool +GetParent() : const SL_Ptr<GR_DLNode> & +SetRRDL(in pRRDL : GR_RRDL*) +GetRRDL() : GR_RRDL * +AddAttrib(in pDLAttrib : SL_Ptr<GR_DLAttrib>, in bAppend : bool = false) : bool +DelAttrib(in pDLAttrib : SL_Ptr<GR_DLAttrib>) +GetAttrib(inout info : const type_info) : SL_Ptr<GR_DLAttrib> +ApplyAttrib(in bApply : bool = true) : bool +GetAttribs() : const GR_DLAttribPtrVect & +IsInvalid() : const bool & +SetTransform(inout mtx : const MH_Matrix44) +GetTransform() : const MH_Matrix44 & +GetNetTransform() : MH_Matrix44 +SetColor(inout clr : const GR_Color) +SetSelect(in eStatus : const EStatus)

GR_DLNode

Class used to stand for some display lists. It is derived from GR_DL.
Specific class can be derived from this class to stand for a collection of points, curves, and

surfaces.

The class diagram is:

GR_DLNode
-m_vDLNode : GR_DLNodePtrVect -m_vDL : GR_DLPtrVect -m_pSelector : GR_Selector *
+Detach() : bool +Render() : bool +GR_DLNode(inout pParentNode : const SL_Ptr<GR_DLNode>, in pSelector : GR_Selector* = 0) +~GR_DLNode() +Invalidate() +Select(inout selSet : GR_SelectionSet, inout filterSet : const GR_SelectionFilterSet, inout ray : const MH_CrvRay, in fTolerance : float) : bool +AddDLNode(inout pDLNode : const SL_Ptr<GR_DLNode>) +RemoveDLNode(inout pDLNode : const SL_Ptr<GR_DLNode>) : bool +AddDL(inout pDL : const SL_Ptr<GR_DL>) +RemoveDL(inout pDL : const SL_Ptr<GR_DL>) : bool +GetChildren(inout pvDLNode : GR_DLNodePtrVect*, inout pvDL : GR_DLPtrVect*)

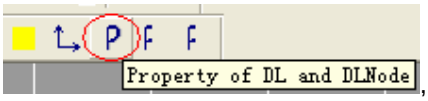
GR_DLAttrib

Class used to stand for a kind of attribute of the display list.

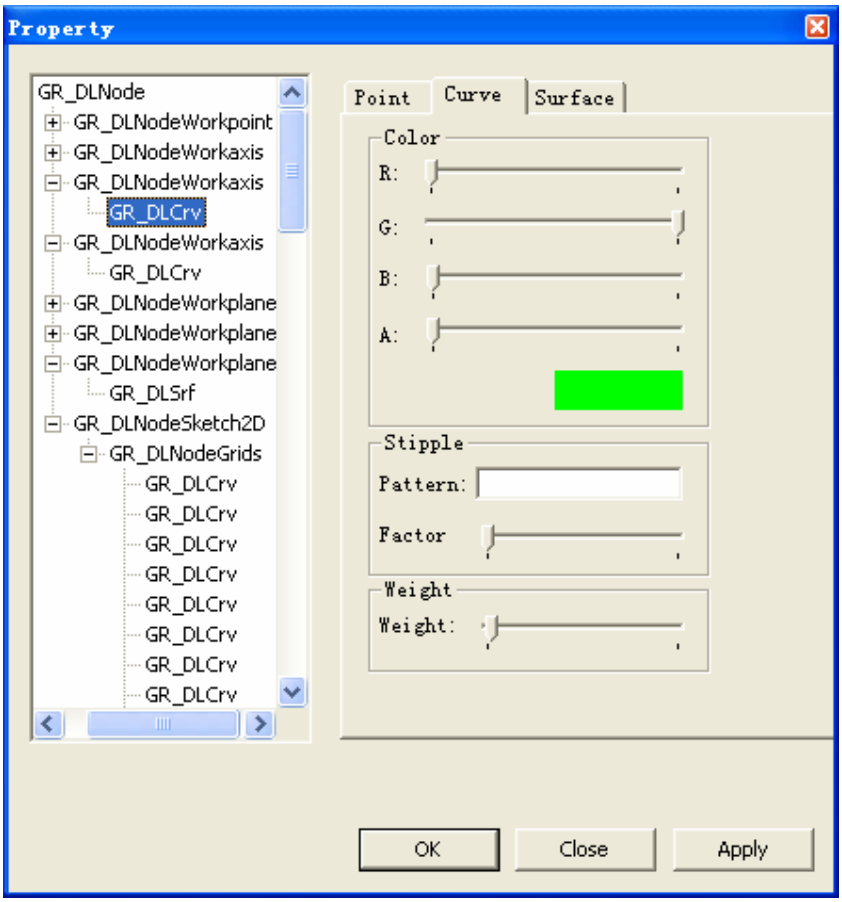
Specific class can be derived from this class to stand for a kind of color, point's size, line's width, etc

The class diagram is:

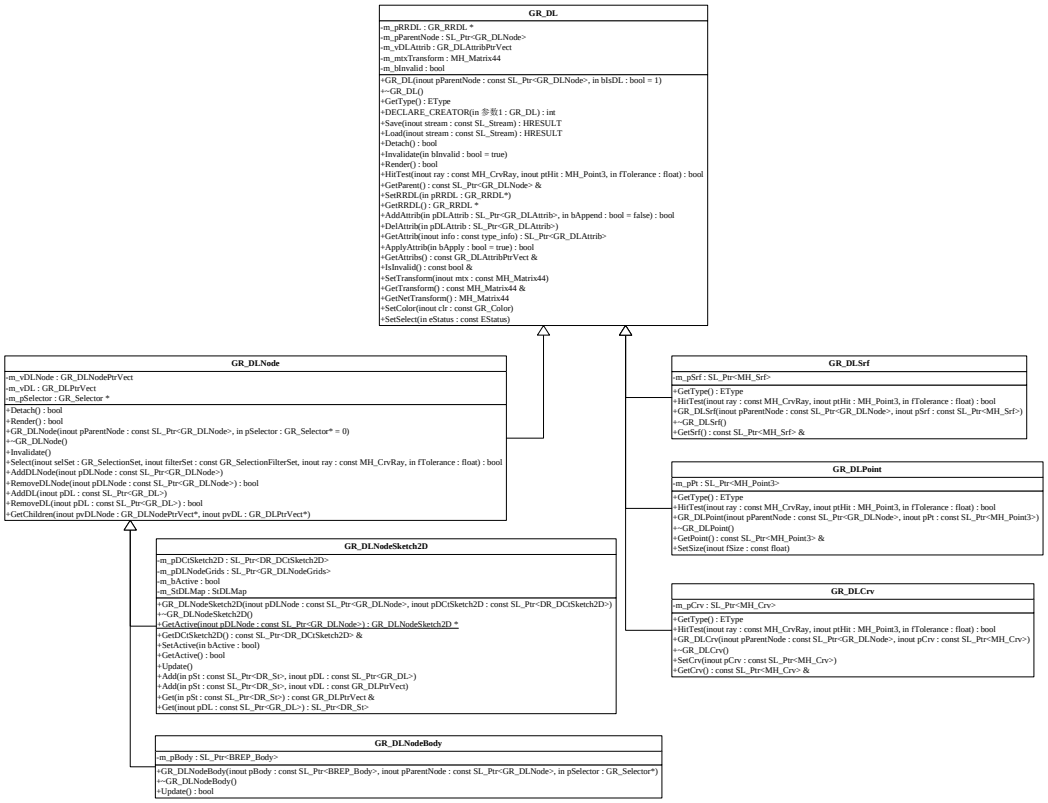
GR_DLAttrib
-m_pRRDLAttrib : GR_RRDLAttrib *
+GR_DLAttrib() +~GR_DLAttrib() +Apply(in bApply : bool) : bool

Using the command "Property of DL and DLNode" 

, we can get the architecture of the display lists and display list nodes as following: (Property of the Y-Axis's display list node)



The hierarchy of the main classes is as below:



Framework library (FW.lib)

The system is a multi-document application.

It consists of some important member variables:

FW_WinApp

This class is derived from CWinApp.

The class diagram is:

FW_WinApp
-m_spIFWWinApp : CComPtr<IFWWinApp> -m_spIFWDocuments : CComPtr<IFWDocuments> -m_bCloseFrame : BOOL -m_pAddInManager : FW_AddInManager * -m_pUIManager : FW_UIManager * -m_pCmdManager : FW_CmdManager * -m_pEventManager : FW_EventManager * -m_pEnvironmentManager : FW_EnvironmentManager *
+FW_WinApp() +~FW_WinApp() +GetWinApp() : FW_WinApp * +GetInterface() : CComPtr<IFWWinApp> +ResetInterface() +GetDocsInterface() : CComPtr<IFWDocuments> +ResetDocsInterface() +InitInstance() : BOOL +ExitInstance() : int +Run() : int +ApcHost_OnCreate() : HRESULT +ApcHost_OpenProject(in bstrProjectFileName : BSTR, in pReferencingProject : IApcProject*, in pProject : IApcProject**) : HRESULT +SetCloseFrame(in bCloseFrame : BOOL) +GetCloseFrame() : BOOL +GetAddInManager() : FW_AddInManager * +GetUIManager() : FW_UIManager * +GetCmdManager() : FW_CmdManager * +GetEventManager() : FW_EventManager * +GetEnvironmentManager() : FW_EnvironmentManager * #OnAppAbout() : void

It contains following managers:

FW_AddInManager

This system supports 3rd-part add-ins. This manager controls the Add-Ins

The class diagram is:

FW_AddInManager
-m_pApp : FW_WinApp * -m_pAddIns : FW_AddIns *
+FW_AddInManager(in pApp : FW_WinApp*) +~FW_AddInManager() +GetAddIns() : FW_AddIns *

FW_CmdManager

This manager controls the commands. We can also execute or terminate the command through this manager.

The class diagram is:

FW_CmdManager
-m_spIFWCmdManager : CComPtr<IFWCmdManager> -m_pApp : FW_WinApp * -m_pCmds : FW_Cmds *
+FW_CmdManager(in pApp : FW_WinApp*) +~FW_CmdManager() +GetInterface() : CComPtr<IFWCmdManager> +ResetInterface() +GetCmds() : FW_Cmds * +Execute(in pCmd : FW_Cmd*) : BOOL +Terminate(in pCmd : FW_Cmd*) : BOOL

FW_UIManager

This manager controls the command's display in the panel bar and toolbar.
The class diagram is:

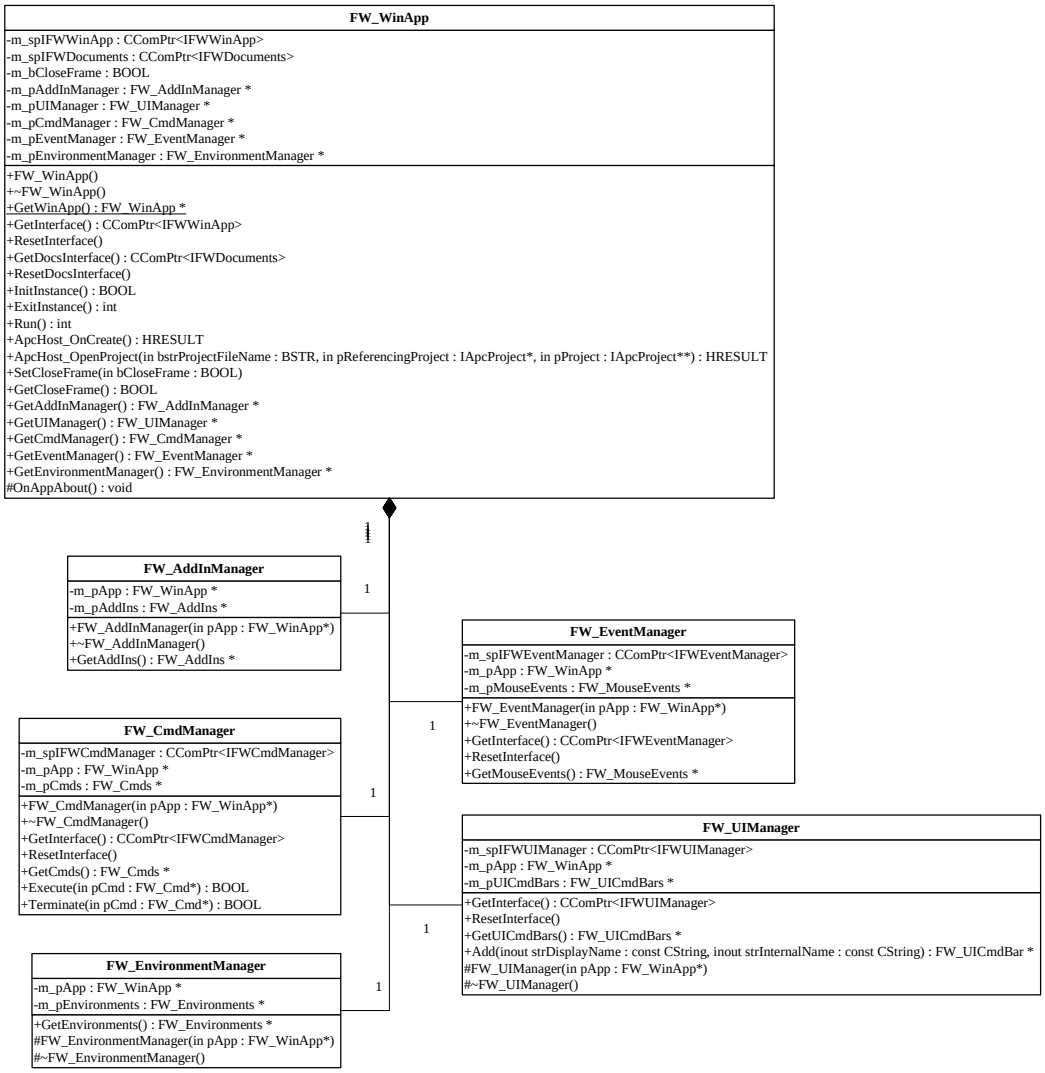
FW_UIManager
-m_spIFWUIManager : CComPtr<IFWUIManager> -m_pApp : FW_WinApp * -m_pUICmdBars : FW_UICmdBars *
+GetInterface() : CComPtr<IFWUIManager> +ResetInterface() +GetUICmdBars() : FW_UICmdBars * +Add(inout strDisplayName : const CString, inout strInternalName : const CString) : FW_UICmdBar * #FW_UIManager(in pApp : FW_WinApp*) #~FW_UIManager()

FW_EventManager

The instance of this class can distribute the events to each command that cares them. So far it can distribute the mouse events. And it is easy to distribute other events that the command cares.
The class diagram is:

FW_EventManager
-m_spIFWEventManager : CComPtr<IFWEventManager> -m_pApp : FW_WinApp * -m_pMouseEvents : FW_MouseEvents *
+FW_EventManager(in pApp : FW_WinApp*) +~FW_EventManager() +GetInterface() : CComPtr<IFWEventManager> +ResetInterface() +GetMouseEvents() : FW_MouseEvents *

The hierarchy of the main classes is listed below:


FW_WinApp
-m_spIFWWinApp : CComPtr<IFWWinApp>
-m_spIFWDocuments : CComPtr<IFWDocuments>
-m_bCloseFrame : BOOL
-m_pAddInManager : FW_AddInManager *
-m_pUIManager : FW_UIManager *
-m_pCmdManager : FW_CmdManager *
-m_pEventManager : FW_EventManager *
-m_pEnvironmentManager : FW_EnvironmentManager *
+FW_WinApp()
+~FW_WinApp()
+GetWinApp() : FW_WinApp *
+GetInterface() : CComPtr<IFWWinApp>
+ResetInterface()
+GetDocsInterface() : CComPtr<IFWDocuments>
+ResetDocsInterface()
+InitInstance() : BOOL
+ExitInstance() : int
+Run() : int
+ApcHost_OnCreate() : HRESULT
+ApcHost_OpenProject(in bstrProjectFileName : BSTR, in pReferencingProject : IApcProject*, in pProject : IApcProject**) : HRESULT
+SetCloseFrame(in bCloseFrame : BOOL)
+GetCloseFrame() : BOOL
+GetAddInManager() : FW_AddInManager *
+GetUIManager() : FW_UIManager *
+GetCmdManager() : FW_CmdManager *
+GetEventManager() : FW_EventManager *
+GetEnvironmentManager() : FW_EnvironmentManager *
#OnAppAbout() : void
1
1
FW_AddInManager
-m_pApp : FW_WinApp *
-m_pAddIns : FW_AddIns *
+FW_AddInManager(in pApp : FW_WinApp*)
+~FW_AddInManager()
+GetAddIns() : FW_AddIns *
1
FW_EventManager
-m_spIFWEventManager : CComPtr<IFWEventManager>
-m_pApp : FW_WinApp *
-m_pMouseEvents : FW_MouseEvents *
+FW_EventManager(in pApp : FW_WinApp*)
+~FW_EventManager()
+GetInterface() : CComPtr<IFWEventManager>
+ResetInterface()
+GetMouseEvents() : FW_MouseEvents *
1
FW_CmdManager
-m_spIFWCmdManager : CComPtr<IFWCmdManager>
-m_pApp : FW_WinApp *
-m_pCmds : FW_Cmds *
+FW_CmdManager(in pApp : FW_WinApp*)
+~FW_CmdManager()
+GetInterface() : CComPtr<IFWCmdManager>
+ResetInterface()
+GetCmds() : FW_Cmds *
+Execute(in pCmd : FW_Cmd*) : BOOL
+Terminate(in pCmd : FW_Cmd*) : BOOL
1
FW_UIManager
-m_spIFWUIManager : CComPtr<IFWUIManager>
-m_pApp : FW_WinApp *
-m_pUICmdBars : FW_UICmdBars *
+GetInterface() : CComPtr<IFWUIManager>
+ResetInterface()
+GetUICmdBars() : FW_UICmdBars *
+Add(inout strDisplayName : const CString, inout strInternalName : const CString) : FW_UICmdBar *
#FW_UIManager(in pApp : FW_WinApp*)
#~FW_UIManager()
1
FW_EnvironmentManager
-m_pApp : FW_WinApp *
-m_pEnvironments : FW_Environments *
+GetEnvironments() : FW_Environments *
#FW_EnvironmentManager(in pApp : FW_WinApp*)
#~FW_EnvironmentManager()
1