SmartPolygonOptimizer™ API

User's manual

Version 1.22

January 12, 2007

3D Incorporated

3D Incorporated LICENSE AGREEMENT

1. Software
As used herein, the term, “Software” means the software accompanying this Agreement, including: (i) the object code form of 3D Incorporated's library of function calls and the ASCII form of 3D Incorporated’s header files for function calls (the “API”); and (ii) the executable form of certain 3D Incorporated's software tools (“Tools”).

2. Evaluation
If you received the Software for the purpose of internal evaluation, as expressed by 3D Incorporated, or if you received the Software without conditions of payment, then the Software is to be used for evaluation purposes only, and this Agreement is effective for a fixed period of time to be determined by 3D Incorporated. If no explicit period of time is given by 3D Incorporated, then this Agreement will terminate in 90 days from receipt of the Evaluation Software, with no written notice of termination required. Upon termination of this agreement, see 9. Term.

3. License Grant
Subject to the terms and conditions of this Agreement, 3D Incorporated grants you a non-exclusive, non-transferable, limited license to: (a) use the copy of the Software and accompanying materials, including a dongle (as applicable to the licensed Software), enclosed in this package (collectively the “Product”) on the Designated System; (b) develop separate software applications derived from the API (the “Applications”); and (c) use, copy, and distribute the Applications; provided, however, that you obtain written approval from 3D Incorporated prior to any sale, license, lease or other distribution of the Applications. You may transfer the Software to the Designated System provided you keep the original Software solely for backup or archival purposes. “Designated Systems” for any Software means a computer system that is: (i) owned or controlled and operated by you; (ii) designated as the computer system on which the Software will be used; and (iii) included a dongle (solely for Software that does not require and unlock code from 3D Incorporated). All rights not expressly granted to you herein are retained by 3D Incorporated. You acknowledge that the Software is copy protected and requires either a key code furnished by 3D Incorporated or an appropriate dongle for continuing operation, as applicable.

4. Software Media and Dongle
You may receive the Product on media which contain various executables or in multiple forms of media. Regardless of the number or types of executables or media you receive, you may use only the media and executables specified in the applicable purchase order or loan agreement. The media may contain executables which have not been licensed; and such unlicensed executables may not be used unless a license is acquired by you from 3D Incorporated. In the event that a dongle that is included as part of the Product you receive with this Agreement is lost or damaged it cannot be replaced by 3D Incorporated, and such loss or damage will require that you purchase another copy of the Software.

5. Ownership
All rights, title and interest to the Product, and any proprietary information contained on the media, or in the associated materials or dongle, are owned by 3D Incorporated and are protected by copyright, trademark and trade secret law and international treaties. You acquire only the right to use the Product during the term of this Agreement. You agree not to develop separate software applications of any kind derived from the Tools or from any other proprietary information of 3D Incorporated, except for the Applications. Any rights, express or implied, in the Product, and any proprietary information contained in the media or dongle other than those specified in this Agreement are reserved by 3D Incorporated. You must treat the Product like any other copyrighted material except as otherwise provided under this Agreement. You agree not to remove, deface or obscure 3D Incorporated’s copyright or trademark notices or legends, or any other proprietary notices in or on the Product or media.

6. Copies and Modifications
You may make one (1) copy of the Software solely for back-up purpose; provided, however, that you reproduce and include all copyright, trademark, and other proprietary rights notices on the copy. You may not make copies of any of the written documentation included in the Product without prior permission, in writing, from 3D Incorporated. You may not nor may you assist another to modify, translate, convert to another programming language, decompile, reverse engineering or disassemble any portions of the Product. Except as otherwise expressly provided by this Agreement, you may not copy the Software. You agree to notify your employees and agents who may have access to the Product of the restrictions contained in this Agreement and to ensure their compliance with such restrictions.

7. Taxes
You shall be liable for and shall pay all charges and taxes, including all sales and use taxes, which may now or hereafter be imposed or levied upon the license or possession or use of the Product, except taxes based on 3D Incorporated’s income.

8. Confidentiality
By accepting this license, you acknowledge that the Product, and any proprietary information contained in the associated media and dongle, are proprietary in nature to 3D Incorporated and contain valuable trade secrets and other proprietary information developed or acquired at

great expense to 3D Incorporated. You agree not to disclose to others or to utilize such trade secrets or proprietary information except as expressly provided herein.

9. Term

This Agreement is effective from the date you use the Software, until the earlier of: (i) the Agreement is terminated; or (ii) if applicable, the dongle is lost or damaged. 3D Incorporated or you may terminate this Agreement at any time by giving thirty (30) days written notice of termination to the other party. Notwithstanding the above, if you fail to comply with any term of this Agreement, or if you become the subject of a voluntary or involuntary petition in bankruptcy or any proceeding relating to insolvency, receivership, liquidation, or composition for the benefit of creditors, if that petition or proceeding is not dismissed with prejudice within thirty (30) days after filling, 3D Incorporated may terminate this Agreement immediately upon notice to you. Promptly upon termination of this Agreement, you agree to cease all use of the Product, and to either destroy or promptly return to 3D Incorporated the Product, together with all copies you made thereof. Notwithstanding the remedies provided above, 3D Incorporated may enforce all of its other legal rights. Sections 4 – 12 and 14 – 18 will survive termination of this Agreement.

10. Assignment

You may not assign, sublicense, rent, loan, lease, convey or otherwise transfer this Agreement, any applicable unlock code, or the Product without prior written permission from 3D Incorporated. Any unauthorized assignment, sublicense, rental, loan, lease, conveyance or other transfer of any copy of the Product or the unlock code shall be void and shall automatically terminate this Agreement.

11. Limited Warranty

3D Incorporated warrants that the Software provided to you shall operate as described in the accompanying documentation under normal use consistent with the terms of this Agreement, for a period of ninety (90) days from the date of your receipt thereof. For the purposes of this Section 10, “Defective Software” means Software which does not operate as described in the accompanying documentation under normal use during the warranty period. 3D Incorporated’s warranty as set forth above shall not be enlarged, diminished or affected by, and no liability shall arise out of, 3D Incorporated’s rendering of technical advice or service in connection with the Product. 3D Incorporated does not warrant that the Software will meet your requirement, operate without interruption or be error free. Your sole remedy under this Section 10 shall be, upon return of the Defective Software to 3D Incorporated, at 3D Incorporeated’s sole discretion: (i) repair or replacement of any Defective Software within the warranty period; or (ii) within the warranty period, return of the amount, if any, paid by you to 3D Incorporated for the Defective Software. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

12. Warranty Exceptions

Except for the warranty expressly provided in Section 10, the Software is provided “as is”. To the maximum extent permitted by applicable law, 3D Incorporated disclaims all other warranties of any kind, express or implied, including, but not limited to, implied warranties of performance, merchantability, and fitness for a particular purpose. You bear all risk relating to quality and performance of the Software, and assume the entire cost of all necessary servicing, repair or correction.

Some jurisdictions do not allow limitations on implied warranties, so the above limitation may not apply to you. In that event, such warranties are limited to the warranty period. This warranty gives you specific legal rights. You may also have other rights which vary from jurisdiction to jurisdiction.

13. Limitation of Remedies

3D Incorporated’s maximum liability for any claim by you or anyone claiming through or on behalf of you arising out of this Agreement shall not in any event exceed the actual amount paid by you for the license to the Product. To the maximum extent permitted by applicable law, 3D Incorporated shall not be liable for the loss of revenue or profits, expense or inconvenience, or for any other direct, indirect, special, incidental, exemplary or consequential damages, arising out of this Agreement or caused by the use, misuse or inability of use the Product, even if 3D Incorporated has been advised of the possibility of such damages. This limited warranty shall not extend to anyone other than the original user of the Product. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

14. Support

3D Incorporated is not responsible for maintaining or helping you to use the Product, and is not required to make available to you any updates, fixes or support for the Product (an “Upgrade”), except pursuant to a separate written Software Maintenance Agreement, except that if any license is included by 3D Incorporated with the upgrade which contains terms additional to or inconsistent with this Agreement, then such additional or inconsistent terms shall supersede the applicable portions of this Agreement when applied to the Upgrade.

15. Governing Law

This Agreement shall be governed by the laws of Japan, exclusive of its choice of law principles.

16. General provisions

If any provision of this Agreement is held to be void, invalid, unenforceable or illegal, the other provisions shall continue in full force and effect. Failure of a party to enforce any provision of this Agreement shall not constitute or be construed as a waiver of such provision or of the right to enforce such provision. If any legal action, including arbitration, arises under this Agreement or by any reason of any asserted breach of this Agreement, the

prevailing party shall be entitled to recover all costs and expenses, including reasonable attomeys' fees, incurred as a result of such legal action.

17. Export

You agree to comply fully with all laws and regulations of Japan and other countries ("Export Laws") to assure that the Product is not: (i) exported, directly or indirectly, in violation of Export Laws; or (ii) used for any purpose prohibited by Export Laws.

18. Acknowledgment

This Agreement is the complete and exclusive statement of agreement between the parties and supersedes all proposals or prior agreements, verbal or written, and any other communications between the parties relating to the subject matter of this Agreement. No amendment to this Agreement shall be effective unless signed by an officer of 3D Incorporated.

SmartPolygonOptimizer™ API

version 1.22

Copyright

©2007. 3D Incorporated. All rights reserved. Made in JAPAN.

Trademarks

SmartCollision, SmartCollision SDK, SmartPolygonOptimizer API are trademarks of 3D Incorporated. Other brand and product names are trademarks of their respective holders.

Web Information

English:
http://www.ddd.co.jp/tech_info/eng_tech_smartcollision.htm

Japanese:
http://www.ddd.co.jp/tech_info/tech_smartcollision.htm

Support

mailto:haptics@ddd.co.jp

Corporate Headquarters

3D Incorporated
http://www.ddd.co.jp/
Urban Square Yokohama 2F,
1-1 Sakae-cho, Kanagawa-ku, Yokohama, 221-0052, Japan
tel:+81-45-450-1330, fax:+81-45-450-1331
mailto:haptics@ddd.co.jp

Contens

Figures

Tables

Lists

1. Preface

SmartPolygonOptimizerAPI is an application interface which can modify and diagnose polygonal models, and was designed to optimize those models especially for use with SmartCollision.

2. Getting Started

2.1 System Requirements

Hardware

General PC

50 MB disk space and 128 MB RAM

Basically no limitation for CPU spec though the faster the better

USB 1.1 or later (for software protection dongle)

Platforms

Microsoft Windows 2000 or XP

Linux (tested on Fedora Core 5, and 6, and Ubuntu 6.10)

Compiler

Microsoft Visual C++ 6.0 or later

gcc (GCC) 4.1.0 or later

2.2 Installation

SmartPolygonOptimizer API does not have the installer program.

You have only to copy all files in the CD-ROM to any place in your PC.

2.3 License Activation

Before run the application that uses SmartPolygonOptimizer API, please insert attached USB key to your PC to activate the license. As for Linux version, please follow the instructions included in the package.

2.4 Files of SDK

2.4.1 Windows version

Figure 2-1 shows the files of SDK. In order to make applications using this API, spo.h must be included in source files of your project and spo.lib must also be linked. At run time, spo.dll is required.

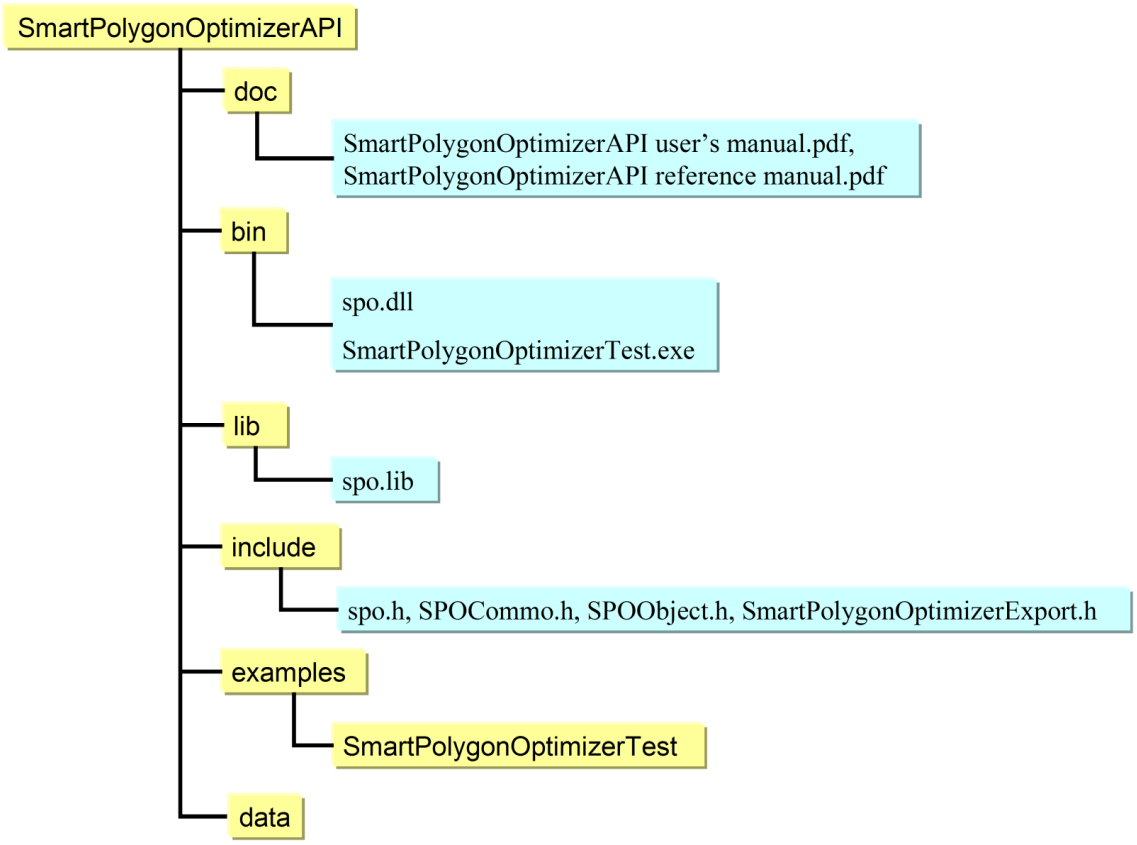


Figure 2-1: Files of API for Windows

2.4.2 Linux version

Figure 2-2 shows the files of SDK. In order to make applications using this API, spo.h must be included in source files of your project and libspo.so must also be linked. At run time, the environment variable LD_LIBRARY_PATH must be set where libspo.so exists. Linux is a directory which stores information or file for Linux users.

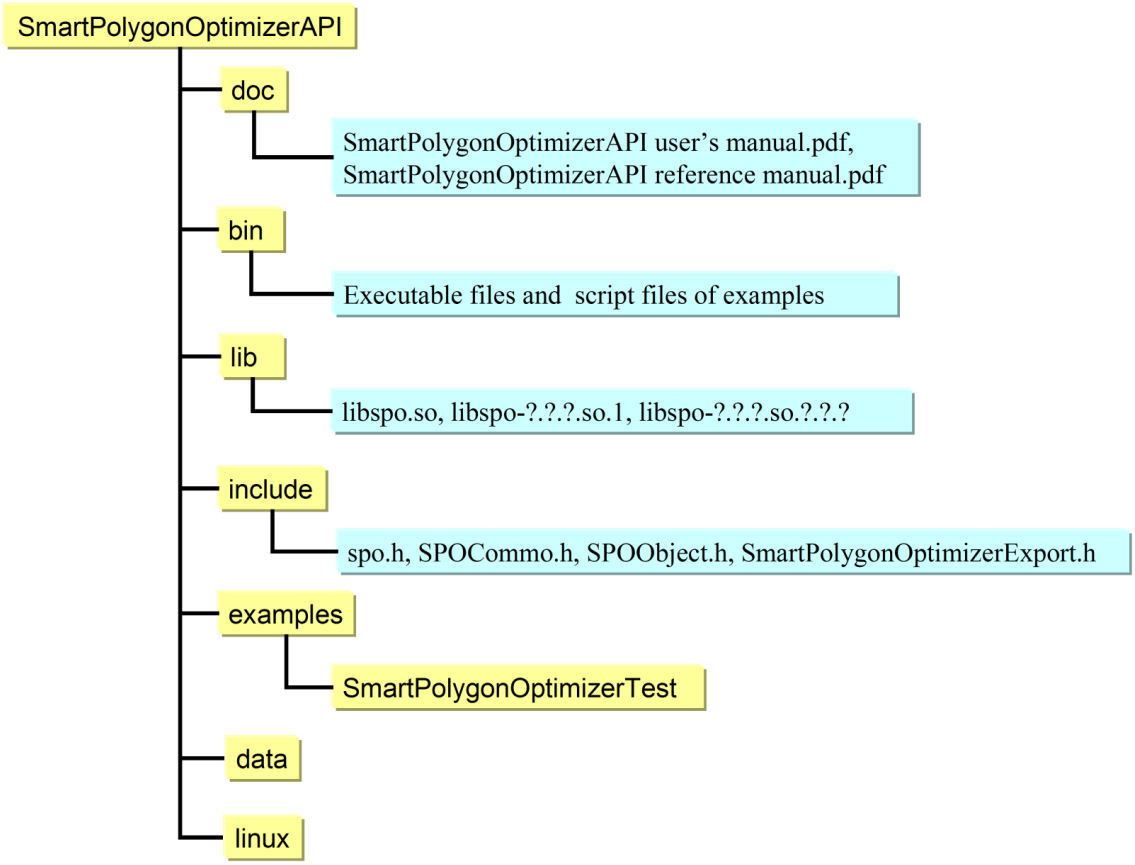


Figure 2-2: Files of API for Linux

3. Class interface

The SmartPolygonOptimizer API consists of a SPOObject class and a SPOPiece struct.

SPOPiece is a struct which stores an indexed triangle set in two dynamically allocated arrays; one array for the vertex data, and one for the index.

SPOObject is a class which stores the SPOPiece objects that together form one object model, and can be directly used by SmartCollision. SPOObject has methods which diagnose various specific features or properties of the triangle set given to it, and can modify the triangle data to fix flaws such as holes, or to optimize for some specific parameter, without changing the model's essential shape.

3.1 How to give geometry data to SPOObject

Geometry data for SPOObject is indexed triangle sets. Figure 3-1 shows the format of indexed triangle set. Indices of vertices start at 0.

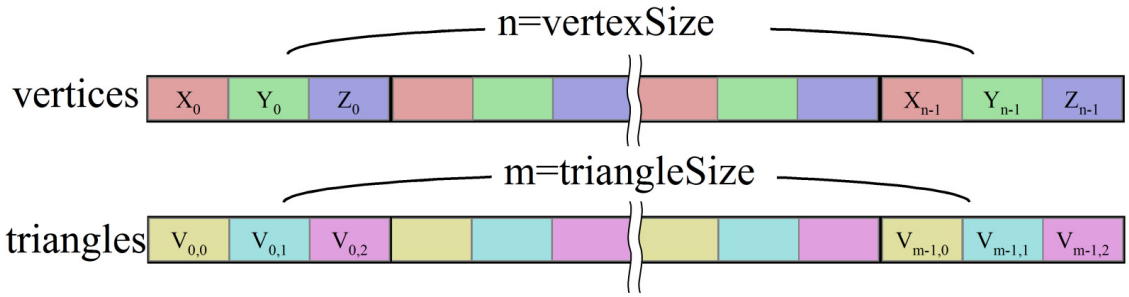


Figure 3-1: Indexed triangle set

The methods to give an indexed triangle set to a SPOObject are as follows.

```
SPOint SPOObject::AddTriangles(const SPOfloat*vertices, SPOint vertexSize,
                               const SPOint*triangles, SPOint triangleSize);
SPOint SPOObject::AddTriangles(const SPOdouble*vertices, SPOint vertexSize,
                               const SPOint*triangles, SPOint triangleSize);
```

An indexed triangle given to SPOObject is treated as one piece. It is possible to call SCObject::AddTriangles multiple times for an object consisting of multiple pieces.

List 3-1 shows how to set geometry of a SPOObject. Figure 3-2 shows the geometry given in List 3-1.

List 3-1: How to set goemetry

```
SPOdouble vertices[3*4]={
    0.0,0.0,0.0, // vertex 0
    1.0,0.0,0.0, // vertex 1
    0.0,1.0,0.0, // vertex 2
    0.0,0.0,1.0  // vertex 3
};
SPOint triangles[3*4]={
    0,2,1, // triangle 0
    1,3,0, // triangle 1
    0,3,2, // triangle 2
    1,2,3  // triangle 3
};

SPOObject object;
If(object.AddTriangles(vertex,4,triangles,4)!=SPO_NO_ERROR){
    // Input geometry is invalid
}
```

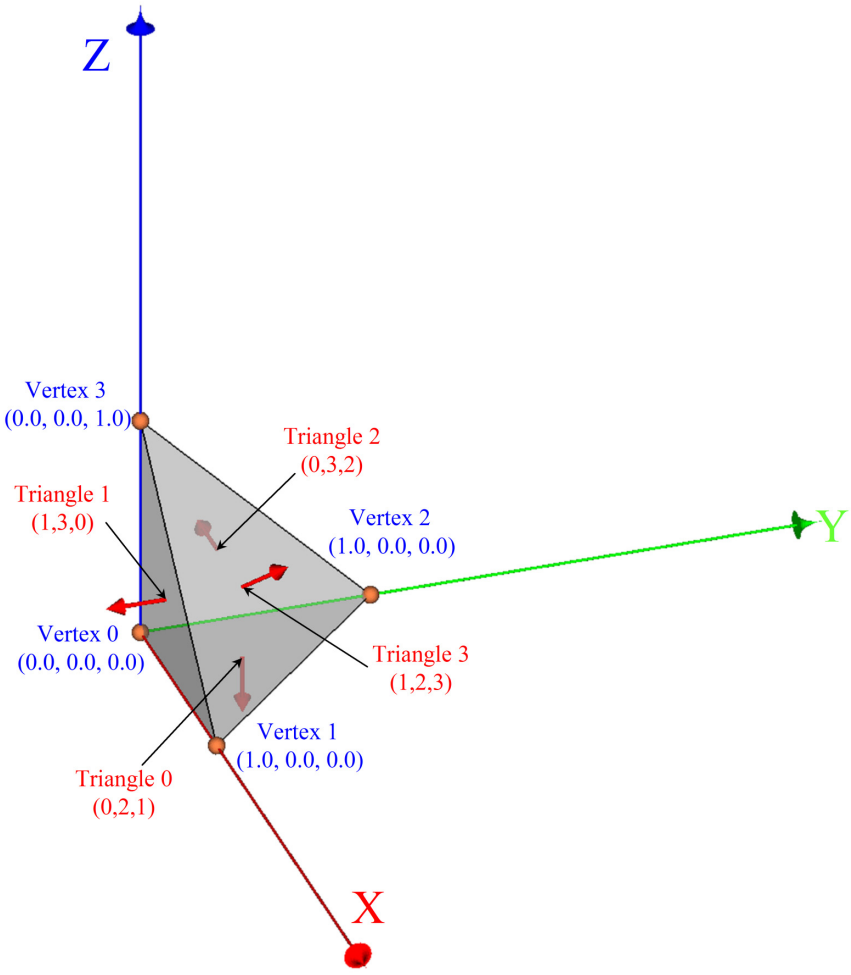


Figure 3-2: An example of geometry

List 3-2 shows how to make an object consisting of multiple pieces.

List 3-2: How to make the object consisting of multiple pieces.

```
SPOObject object

If(object.AddTriangles(vertex1,vertexCount1,triangles1,triangleCount1)!=SC_NO_ERROR){
    // Input geometry is invalid
}
If(object.AddTriangles(vertex2,vertexCount2,triangles2,triangleCount2)!=SC_NO_ERROR){
    // Input geometry is invalid
}
If(object.AddTriangles(vertex3,vertexCount3,triangles3,triangleCount3)!=SC_NO_ERROR){
    // Input geometry is invalid
}
If(object.AddTriangles(vertex4,vertexCount4,triangles4,triangleCount4)!=SC_NO_ERROR){
    // Input geometry is invalid
}
```

3.2 Definition of SPOPieces

The definition of SPOPiece is as follows.

List 3-3: Definition of SPOPiece

```
struct SPOPiece{
	SPOreal*vertices;
	SPOint vertexSize;
	SPOint*triangles;
	SPOint triangleSize;
};
```

The type of SPOreal is either float or double, depending on the version of SmartPolygonOptimizer.

List 3-4 shows how to get the number of SPOPieces within a SPOObject, and pointers to each SPOPiece.

List 3-4: How to access SPOPiece struct

```
int pieceCount=object.GetPieceCount();// Get the number of pieces
for(i=0;i<pieceCount;i++){
	const SPOPiece*p=object.GetPiece(i);
	//
}
```

3.3 Category of methods of SPOObject

The methods of SPOObject are categorized as shown in Figure 3-1.

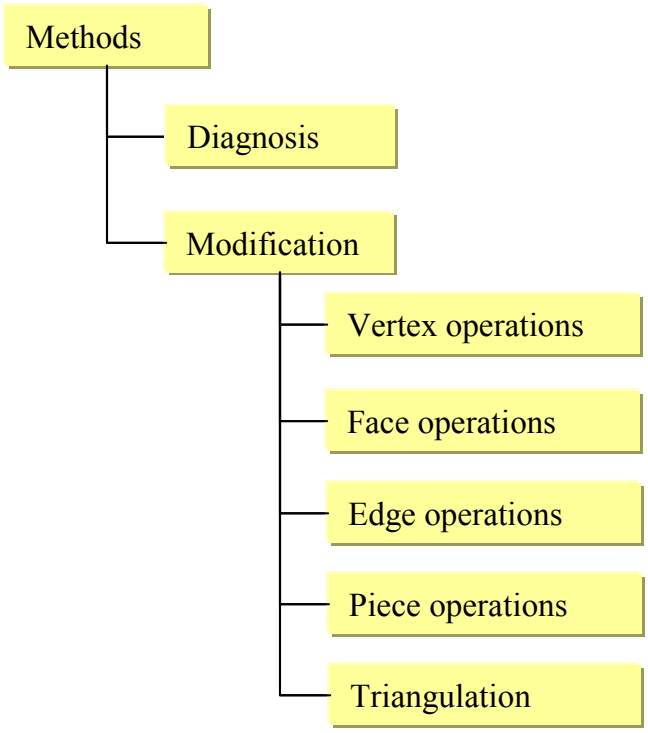


Figure 3-1: Categorization of methods of SPOObject

4. Diagnosis methods

These methods give the user useful information about the nature of the triangle data within an SPOObject. Figure 4-1 gives definitions of a triangle and its elements, such as edges, vertices, and normal that will be used in describing these methods.

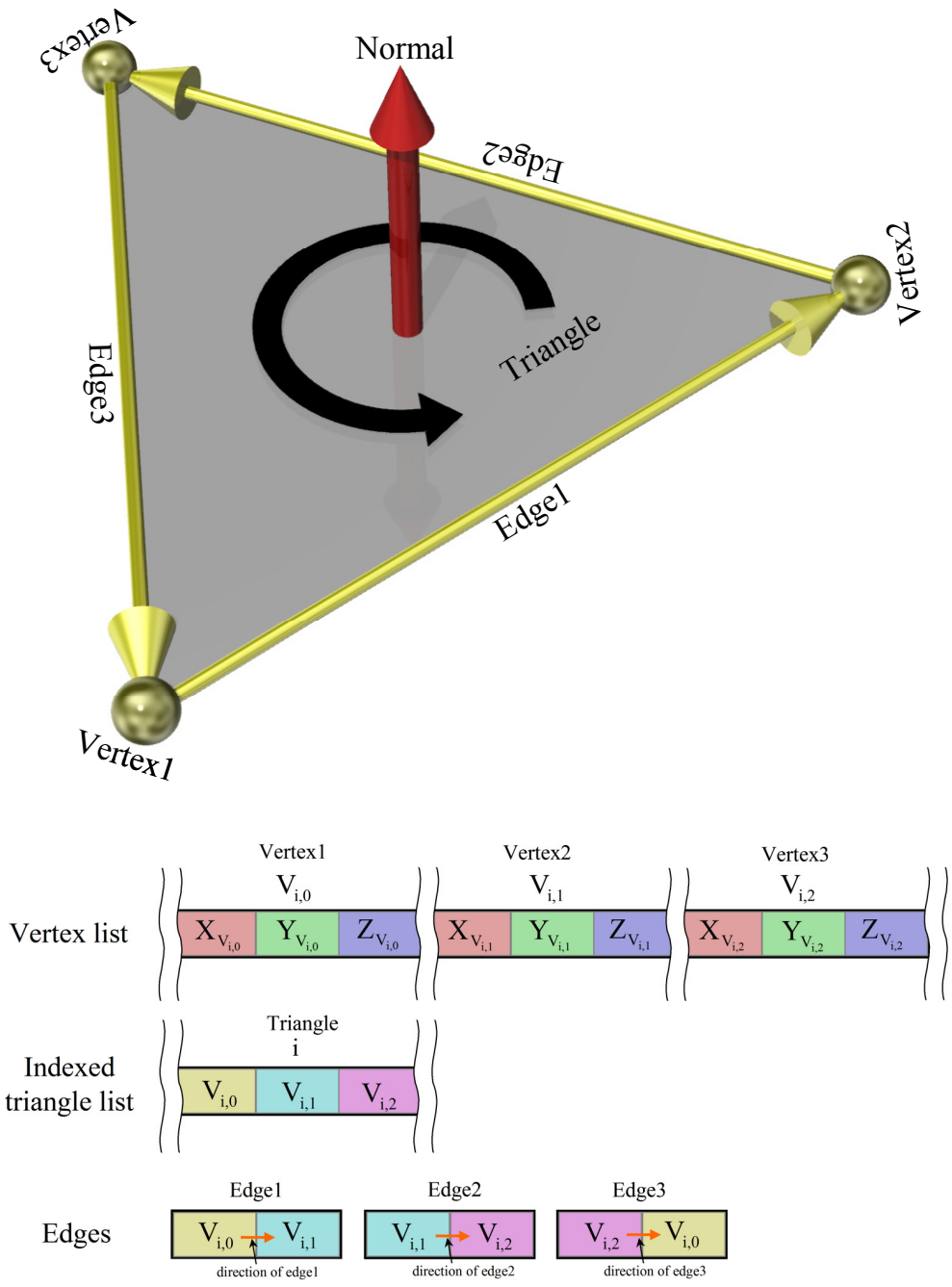


Figure 4-1: Definition of a triangle and its elements.

4.1 Types of edges in terms of connectivity

Table 4-1 and Figure 4-2 show types of edges classified by connectivity. Figure 4-3 shows examples of those types of edges. Edge direction is given by vertex winding, which is determined by the order in which the vertices are specified in the triangle-indexed list of the SPOPeice as shown in Figure 4-1.

Table 4-1: Types of edge classified by its connectivity

Name	Type	Description
Linked edge	SPO_EDGE_TYPE_LINKED	An edge which shares at least one other edge wound in the opposite direction.
Unlinked edge	SPO_EDGE_TYPE_UNLINKED	An edge which shares no other edge wound in the opposite direction.
Duplicate edge	SPO_EDGE_TYPE_UNLINKED	An unlinked edge which shares at least one edge wound in the same direction.
Branched edge	SPO_EDGE_TYPE_BRANCHED	A linked edge which shares at least one edge in the opposite direction and at least one edge in either direction.

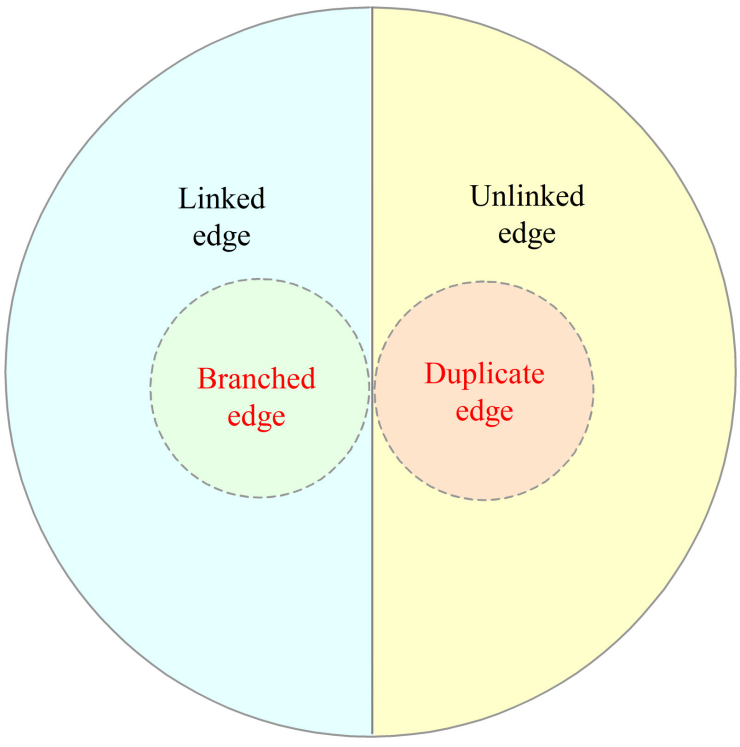
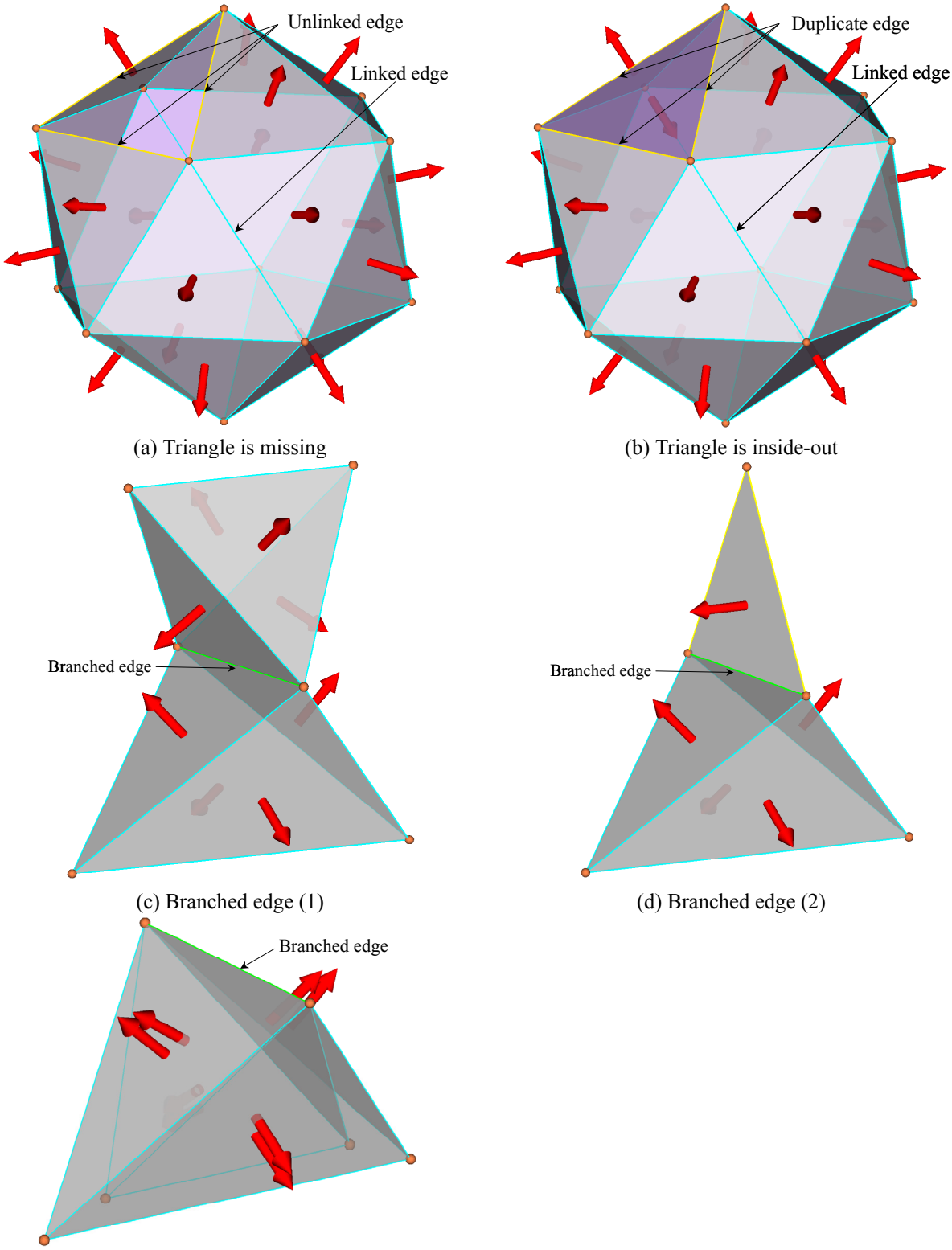


Figure 4-2: A set diagram of types of edges



(a) Triangle is missing

(b) Triangle is inside-out

(c) Branched edge (1)

(d) Branched edge (2)

(e) Branched edge (3)

Figure 4-3: Examples of various types of edge (by connectivity)

4.2 Types of edges in terms of covexity

Linked edges can also be classified by its convexity. Convexity of edges can be defined by the angle θ of the edge as shown in Figure 4-4. The value of θ can be between $-\pi$ and π.

$$\begin{cases} \cos\theta = \boldsymbol{n}_1 \cdot \boldsymbol{n}_2 \\ \sin\theta = (\boldsymbol{n}_1 \times \boldsymbol{n}_2) \cdot \dfrac{\boldsymbol{e}_1}{|\boldsymbol{e}_1|} \end{cases}$$

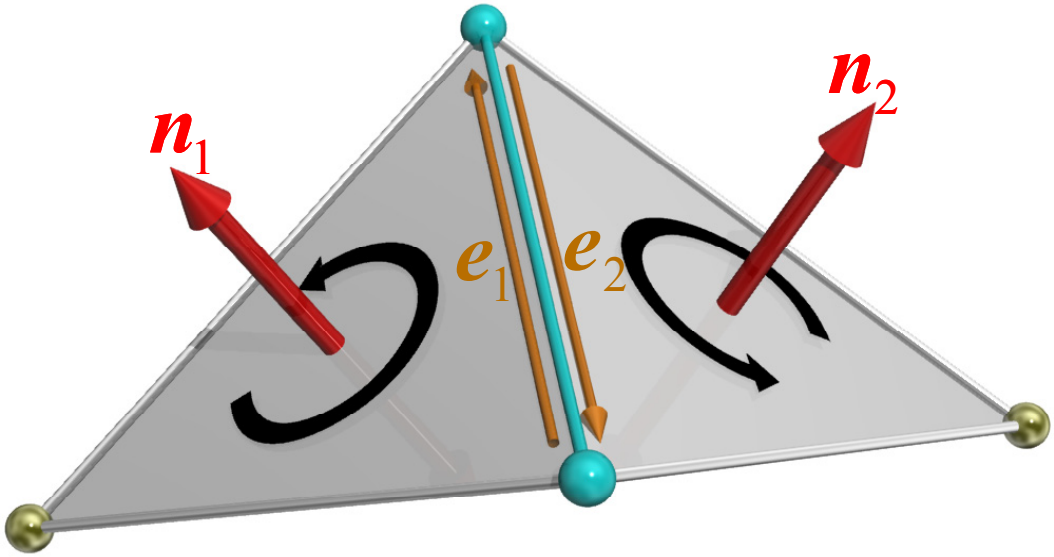


Figure 4-4: Definition of the angle of the edge

Edges are classified by its angle θ and a tolerance value ε as shown in Table 4-2, Figure 4-5 and Figure 4-6.

Table 4-2: Types of edge classified by its convexity

Name	Type	Description
convex edge	SPO_EDGE_TYPE_CONVEX	The edge, whose angle θ is greater than ε.
conave edge	SPO_EDGE_TYPE_CONCAVE	The edge, whose angle θ is less than ε.
Flat edge	SPO_EDGE_TYPE_FLAT	The edge, whose absolute value of angle θ is less than ε.
Folded edge	SPO_EDGE_TYPE_FOLDED	The edge, whose angle θ is less than $-\pi+\varepsilon$ and greater than $\pi-\varepsilon$.

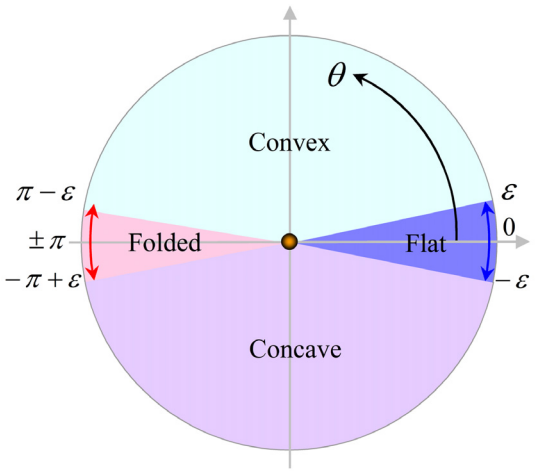


Figure 4-5: Relation between angle and type of edge

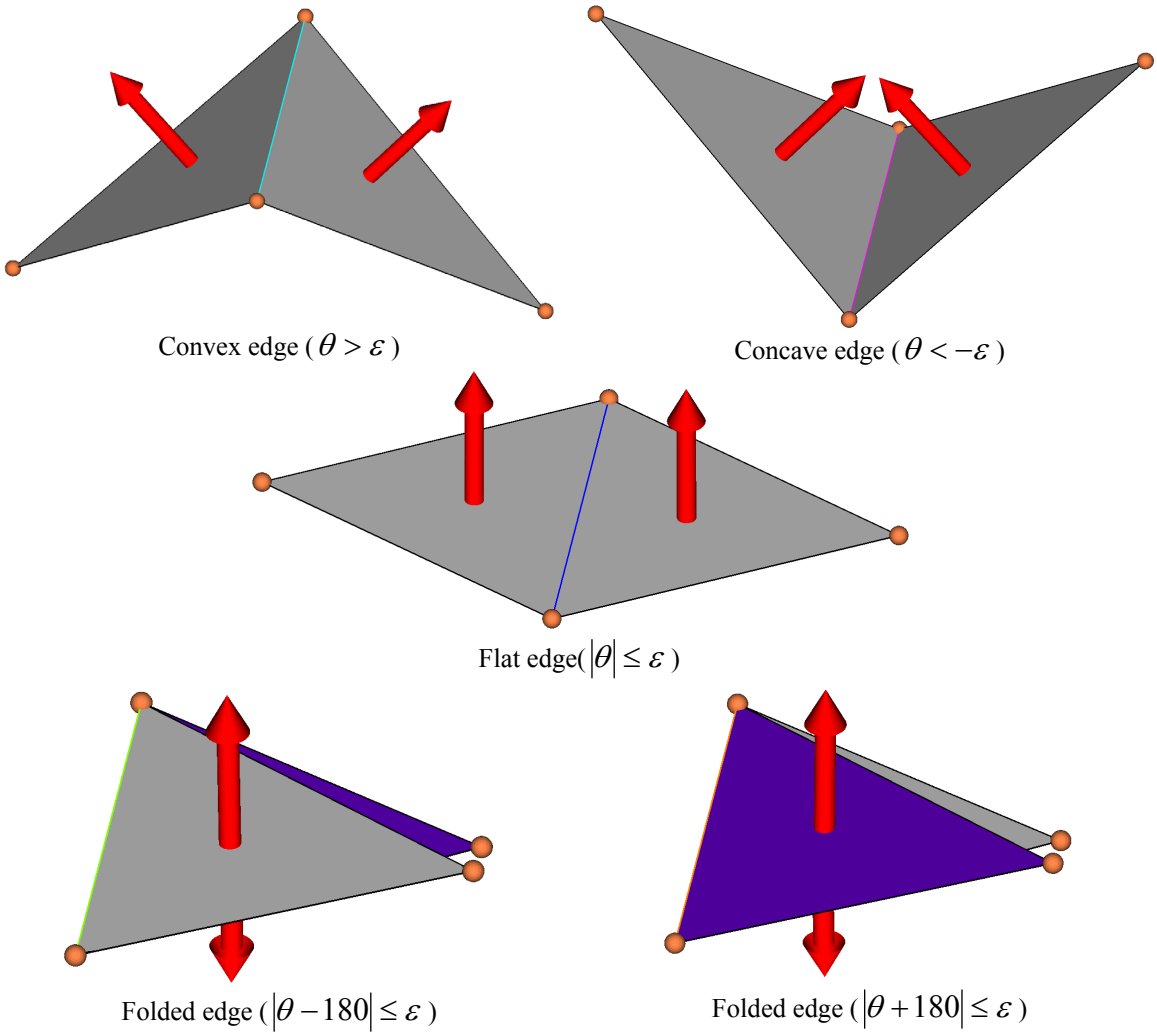


Figure 4-6: Types of edge classified by its angle

4.3 GetEdgeCount

This method counts the number of specific type of edges. Types of edge are shown in Table 4-1 and Table 4-2.

```
SPOint SPOObject::GetEdgeCount(SPOenum type, SPOdouble tolerance);
SPOint SPOObject::GetEdgeCount(SPOenum type, int index,SPOdouble tolerance);
```

List 4-1: How to count the number of specific type of edges

```
SPOObject object;
// Setup of geometry
…
SPOdouble tolerance=1e-2;// tolerance about angle of edge [degree]
SPOint linkedEdgeCount;
SPOint unlinedEdgeCount;
SPOint duplicateEdgeCount;
SPOint branchedEdgeCount;

SPOint convexEdgeCount;
SPOint concaveEdgeCount;
SPOint flatEdgeCount;
SPOint foldingEdgeCount;

linkedEdgeCount  =object.GetEdgeCount(SPO_EDGE_TYPE_LINKED,tolerance);
unlinkedEdgeCount=object.GetEdgeCount(SPO_EDGE_TYPE_UNLINKED,tolerance);
duplicateEdgeCount=object.GetEdgeCount(SPO_EDGE_TYPE_DUPLICATE,tolerance);
BranchedEdgeCount=object.GetEdgeCount(SPO_EDGE_TYPE_BRANCHED,tolerance);

linkedEdgeCount  =object.GetEdgeCount(SPO_EDGE_TYPE_CONVEX,tolerance);
unlinkedEdgeCount=object.GetEdgeCount(SPO_EDGE_TYPE_CONCAVE,tolerance);
duplicateEdgeCount=object.GetEdgeCount(SPO_EDGE_TYPE_FLAT,tolerance);
BranchedEdgeCount=object.GetEdgeCount(SPO_EDGE_TYPE_FOLDED,tolerance);
```

4.4 IsClosed

This method diagnoses whether the object is closed or not. Closed objects must not have unlinked edges.

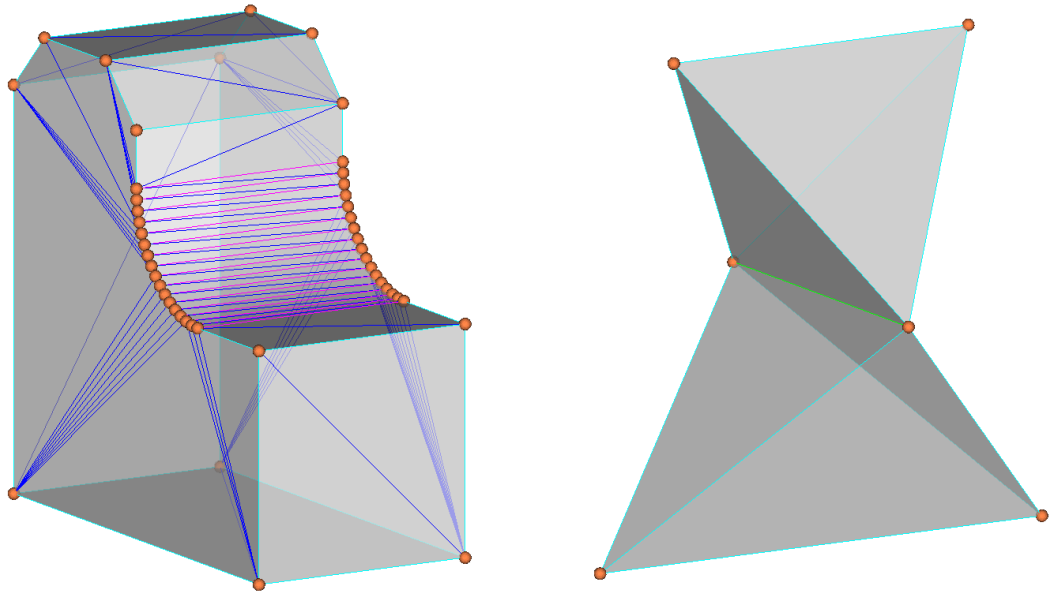
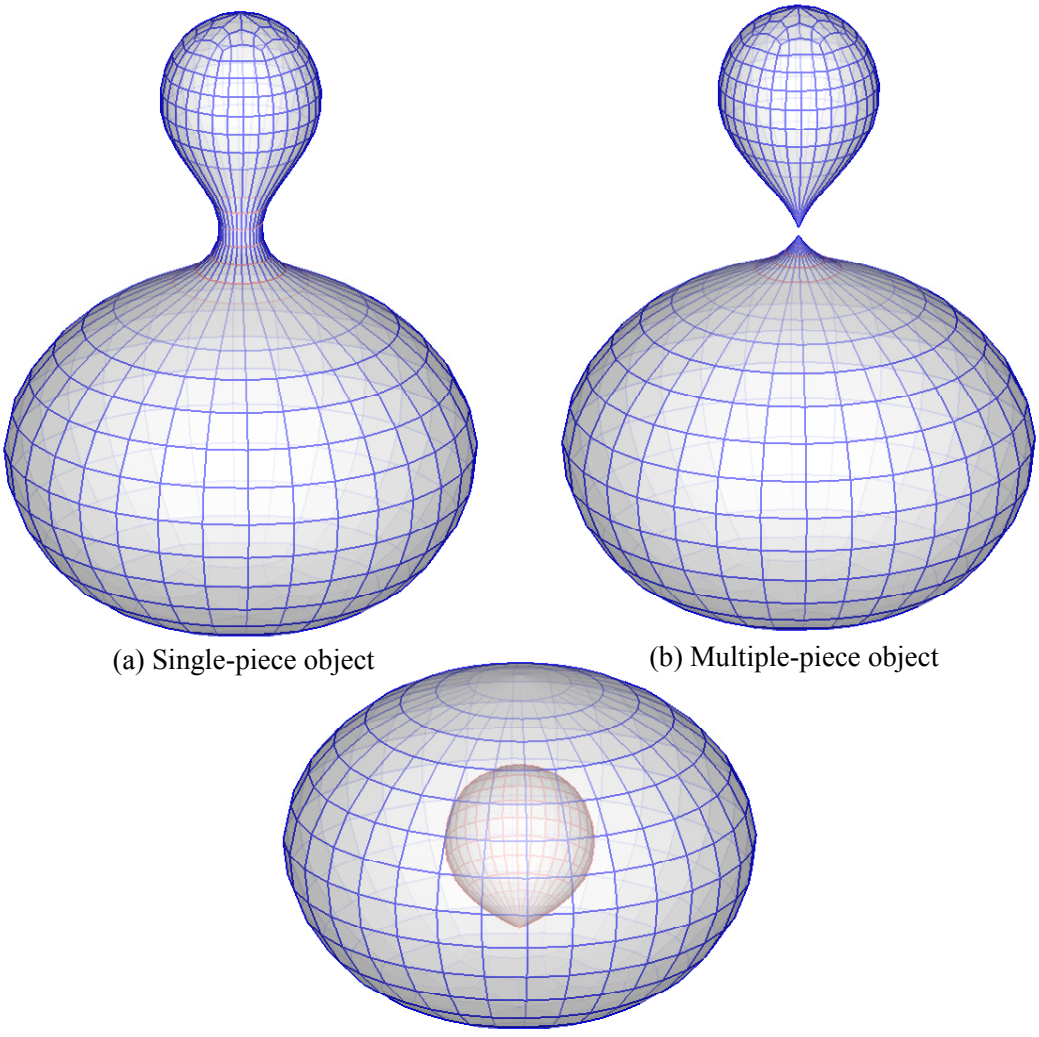


Figure 4-7: Examples of closed objects

4.5 IsSingleBoundary

This method diagnoses whether the objects have a single-boundary or not. A single-boundary object consists of **single-boundary pieces**, and a single-boundary piece is surrounded by only one boundary. On the other hand, a **multiple-boundary piece** is surrounded by multiple boundaries. Figure 4-8 (a) shows an example of single boundary piece and Figure 4-8 (b) shows an example of multiple-boundary piece. Figure 4-8 (c) shows another type of multiple-boundary piece which has empty space inside.



(a) Single-piece object

(b) Multiple-piece object

(c) Multiple-piece object(hollow object)

Figure 4-8: Single/Multiple-boundary piece

If an object has at least one branched edge, it might be a multiple-boundary piece. Therefore, such kinds of objects are not considered as being single-boundary.

4.6 IsConvex

This method diagnoses whether the objects is convex or not. Figure 4-9 shows examples of convex objects. The objects, which have only convex edges and flat edges are classified as a **convex object**. The objects, which have only concave edges and flat edges are also classified as convex objects. The objects, which have only folded edges and flat edges are a **flat object**. Convexity of flat objects can be determined by the convexity of vertices on folded edges as shown in Figure 4-10. If there is at least a concave vertex, the object is not convex.

On the other hand, the object which is not a convex object is classified as a **non-convex object**. Figure 4-11 shows examples of non-convex objects.

If the object has unlinked edges, this method return false.

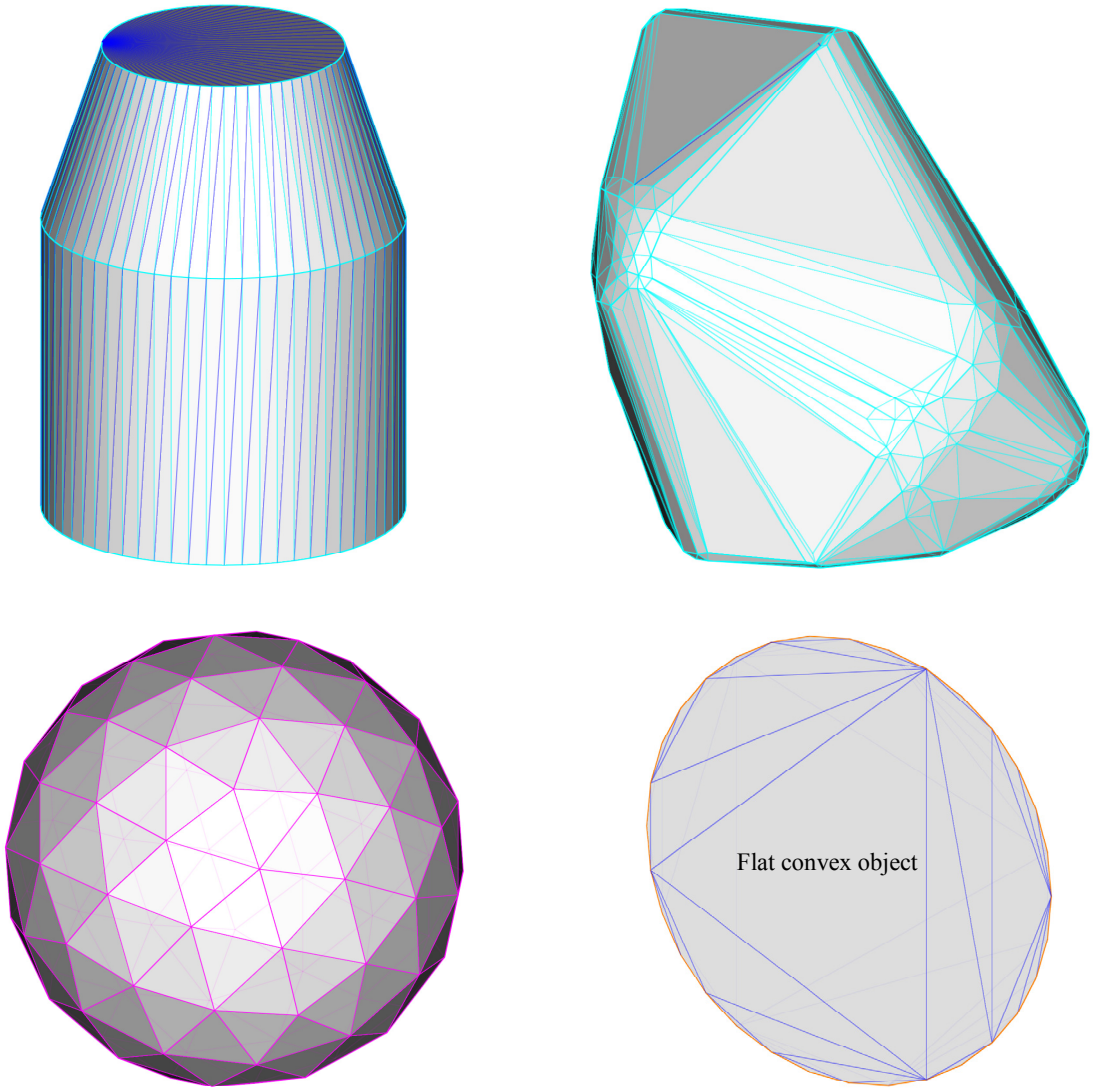


Figure 4-9: Examples of convex objects

$$\text{Angle between folded edges} \begin{cases} \cos\theta = \dfrac{\boldsymbol{e}_1 \cdot \boldsymbol{e}_2}{|\boldsymbol{e}_1||\boldsymbol{e}_2|} \\ \sin\theta = \dfrac{\boldsymbol{e}_1 \times \boldsymbol{e}_2}{|\boldsymbol{e}_1||\boldsymbol{e}_2|} \cdot \boldsymbol{n}_1 \end{cases} \quad \begin{cases} \text{if } \theta > 0 \text{ then convex vertex} \\ \text{if } |\theta| <= \varepsilon \text{ then flat vertex} \\ \text{if } \theta < 0 \text{ then concave vertex} \\ \text{if } |\pm\theta \mp \varepsilon| > 0 \text{ then folded vertex} \end{cases}$$

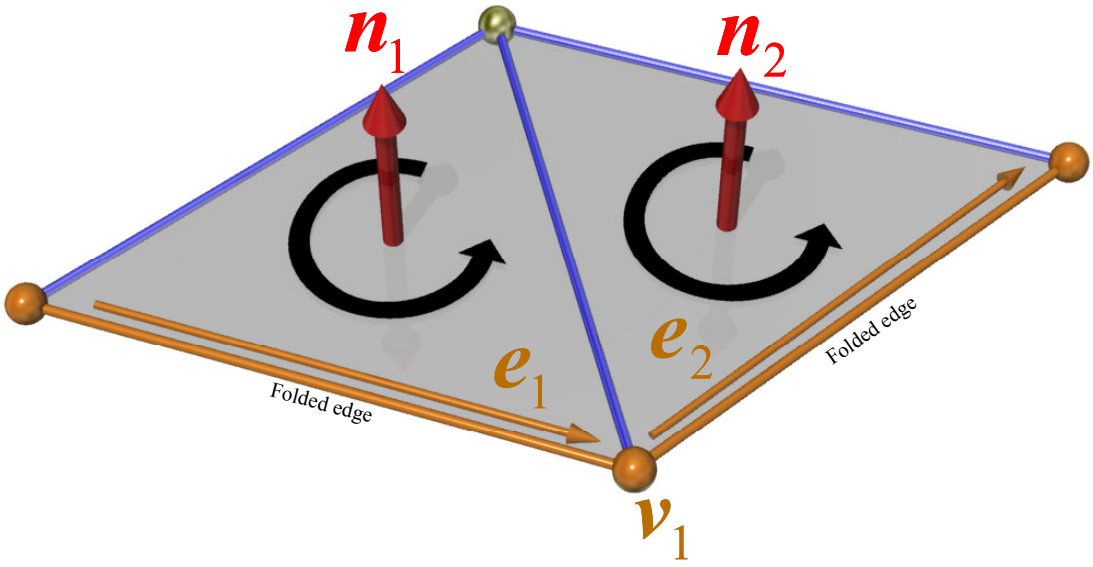


Figure 4-10: Convexity of a vertex on folded edges in a flat object

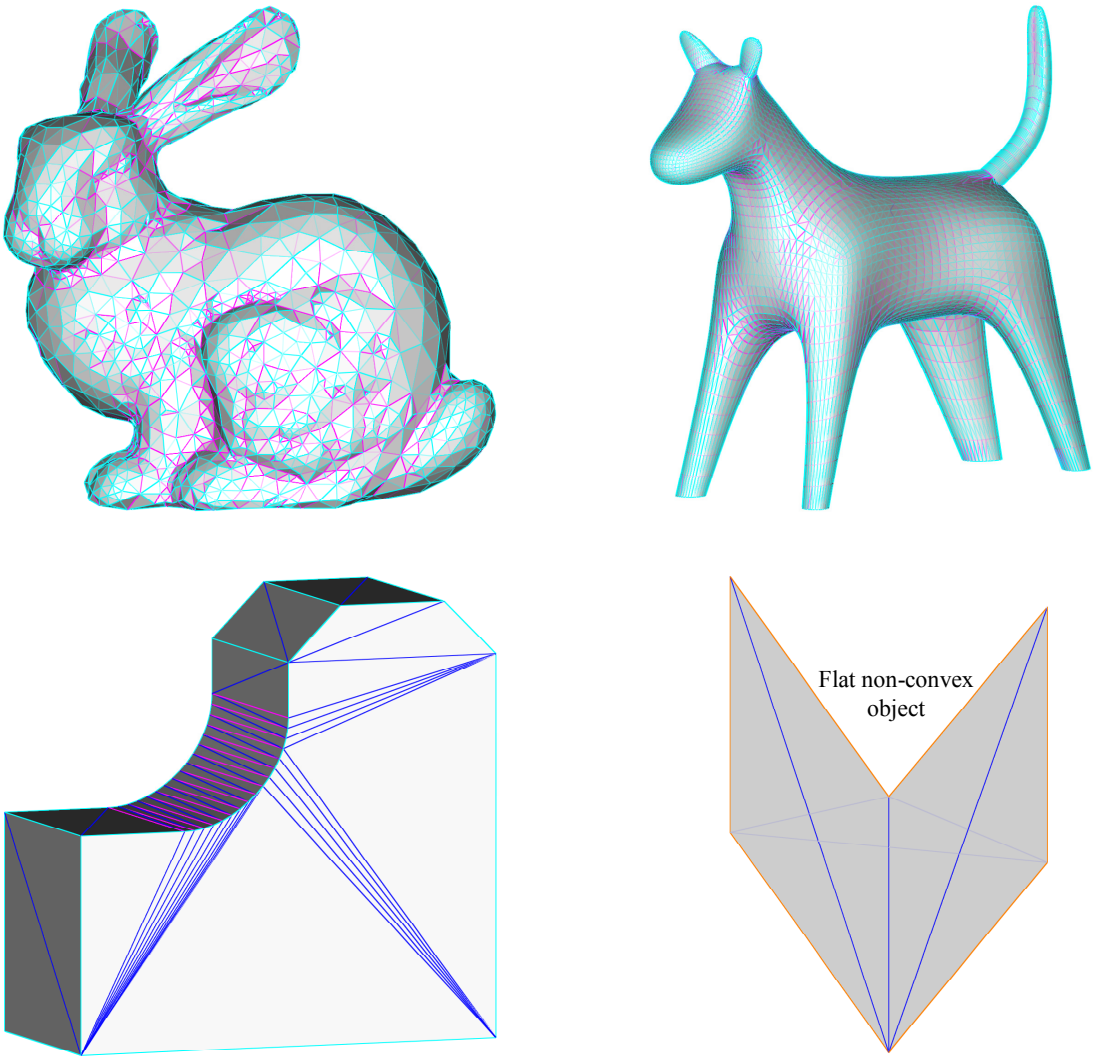


Figure 4-11: Examples of non-convex objects

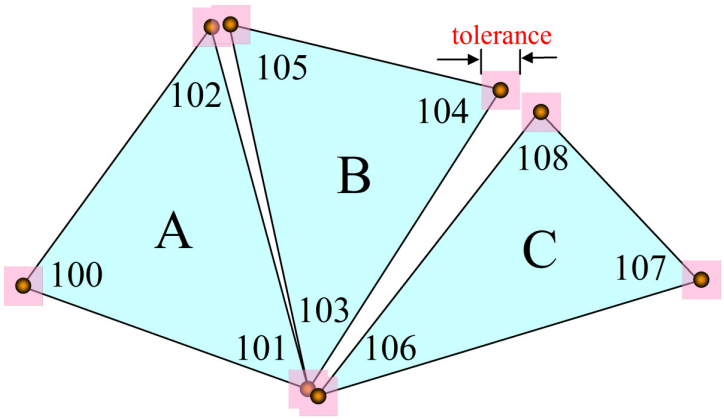
5. Modification methods

These methods allow the user to modify the triangle data in an SPOObject in order to either correct flaws in the input data, or to optimize the data for a particular purpose, all without changing the essential shape of the model.

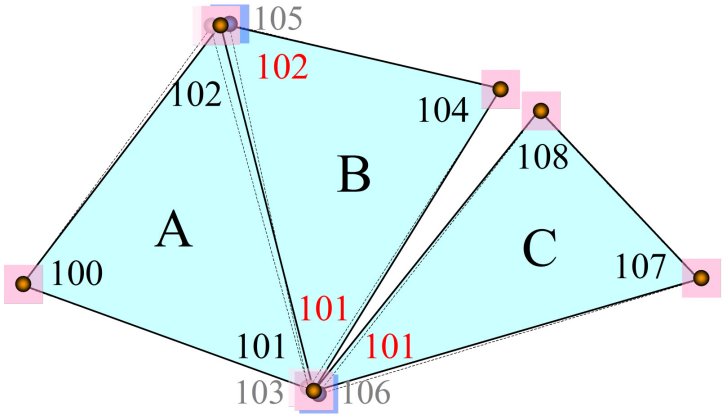
5.1 Vertex operations

5.1.1 ConnectVertices

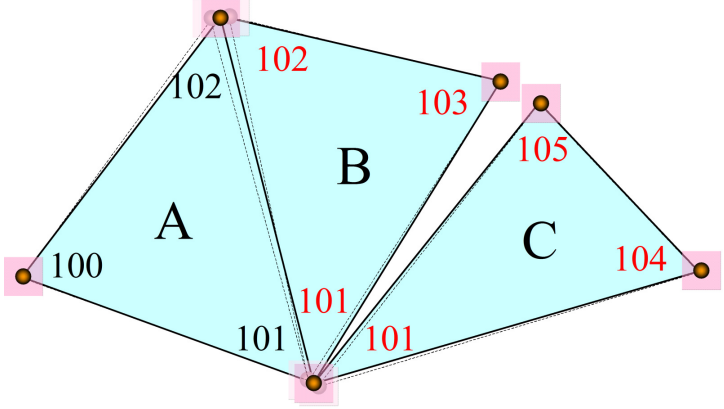
This method connects vertices of which the difference between each coordinate is under a given tolerance. This method also removes degenerate triangles.



(a) Before processing



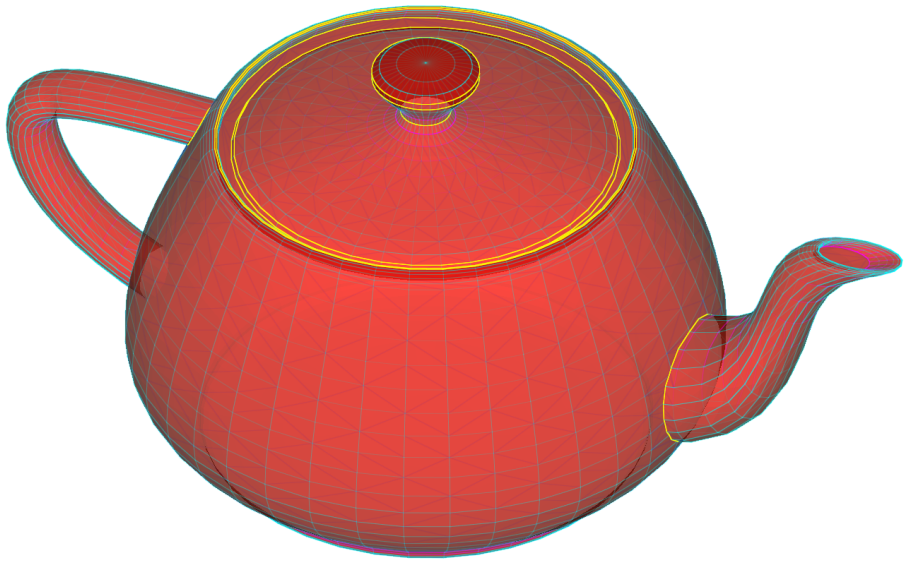
(b) After processing(removeVertexFlag=false)



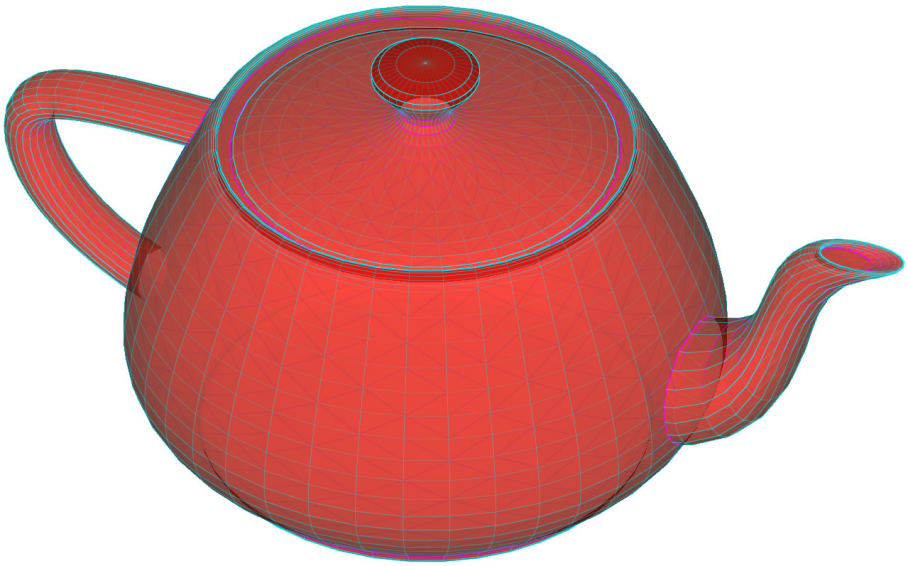
(b) After processing(removeVertexFlag=true)

Figure 5-1 : Examples of ConnectVertices(1)

In Figure 5-2(a), there are duplications of vertices in yellow edges. In Figure 5-2(b), duplications of vertices are removed.



(a)Before processing



(b)After processing

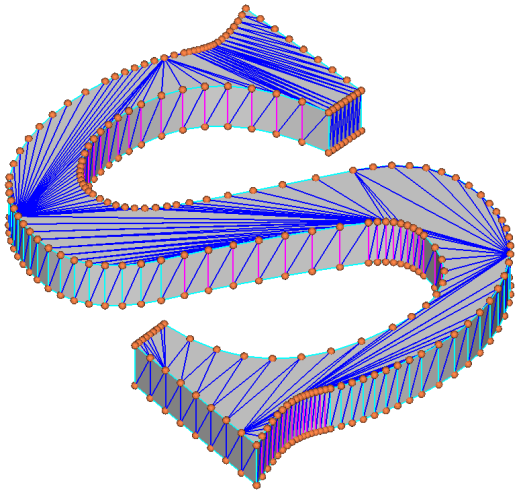
Figure 5-2 : Examples of ConnectVertices(2)

5.1.2 RemoveRedundantVertices

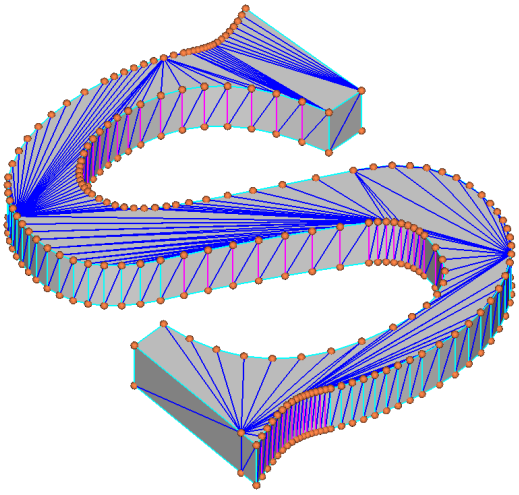
This method removes redundant vertices.

This method is closely related to triangulation (See section 5.5). Figure 5-3 (a),(b),(c) shows examples of RemoveRedundantVertices. Even if there are potentially redundant vertices, particular triangulation patterns might not allow them to be removed. Therefore, changing the triangulation pattern may help to remove more vertices.

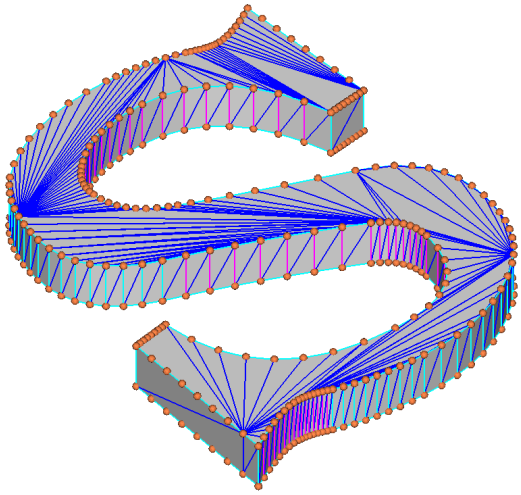
Figure 5-3(d) shows the result of RemoveRedundantVertices in combination with ChangeTriangulationPattern.



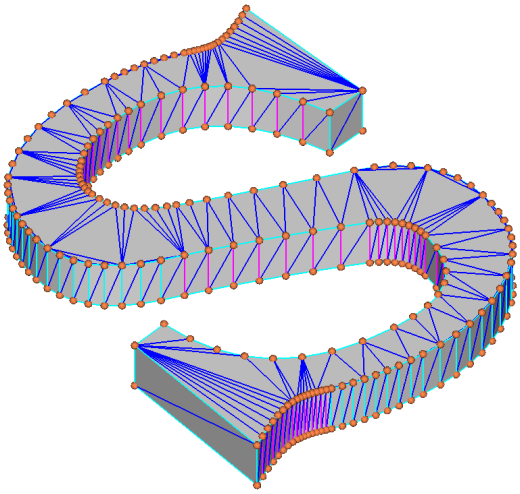
(a)Before processing



(b)After processing(removeVerticesFlag=true)



(c)After processing(removeVerticesFlag=false)



(d)After processing(removeVerticesFlag=true) in combination with ChangeTriangulationPattern

Figure 5-3：Examples of RemoveRedundantVertices

5.2 Face operations

5.2.1 RemoveThinTriangles

This method removes thin triangles from objects and their pieces. Thin triangles are defined as the triangles whose width are less equal than tolerance. Width of triangle(W) is defined as shown in Figure 5-4.

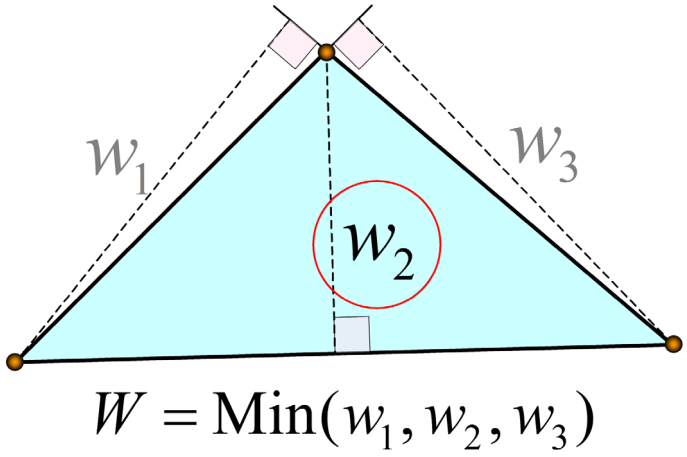
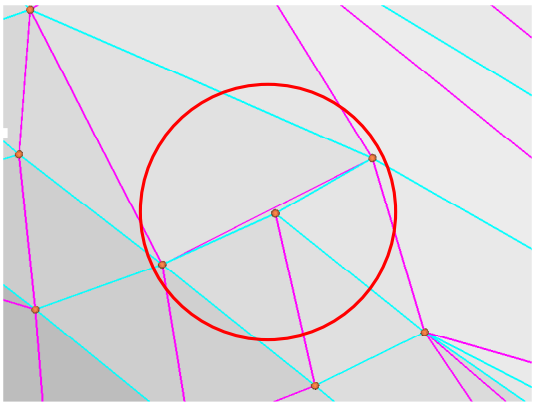
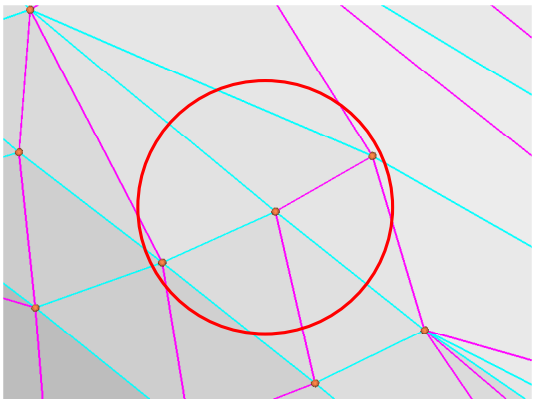


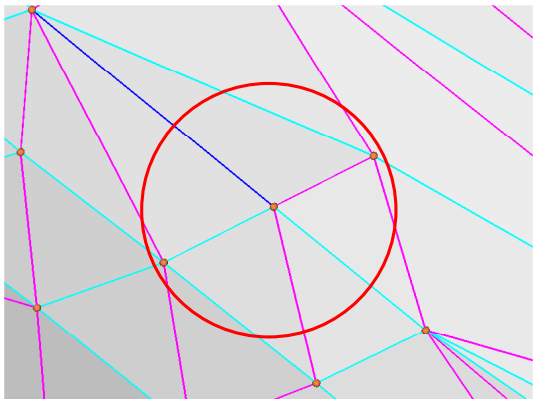
Figure 5-4: Width of a triangle



(a) Before processing



(b) Before processing(moveVerticesFlag=false)



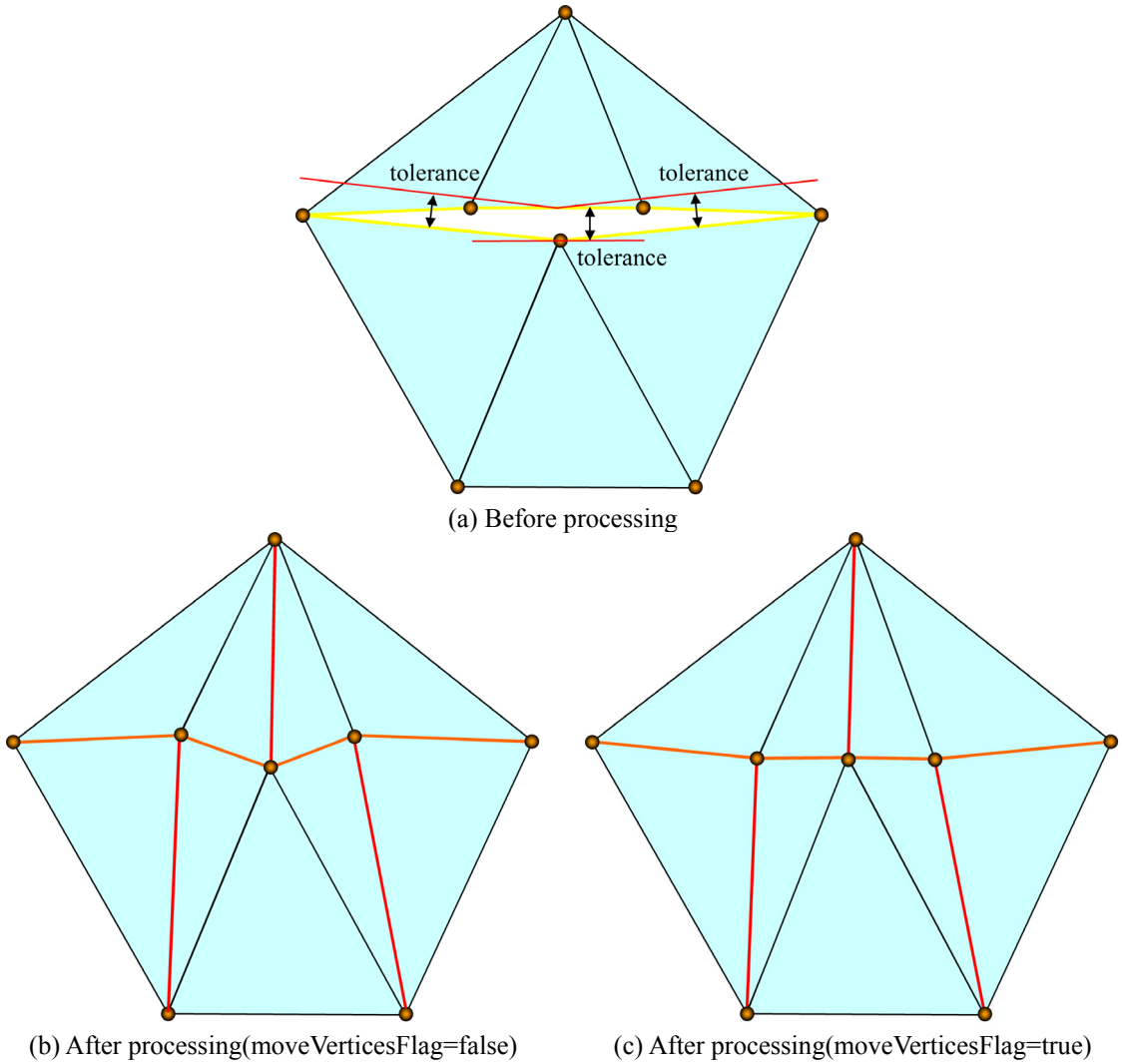
(c) Before processing(moveVerticesFlag=true)

Figure 5-5: Examples of RemoveThinTriangles

5.3 Edge operations

5.3.1 SplitEdges

This method splits edges which has vertices which are nearer than a given tolerance from other edges.

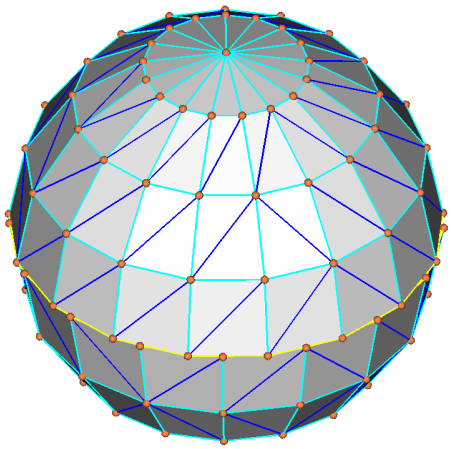


(a) Before processing

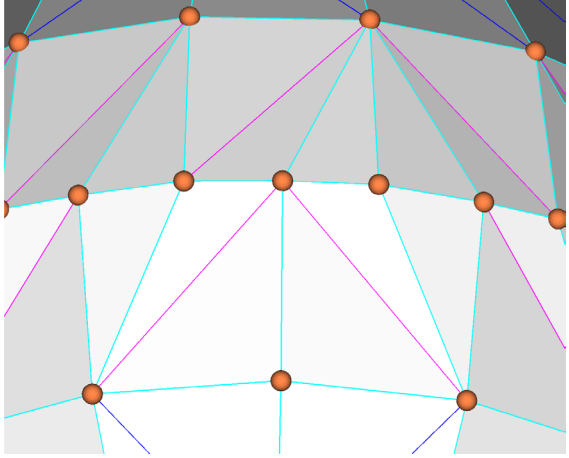
(b) After processing(moveVerticesFlag=false)

(c) After processing(moveVerticesFlag=true)

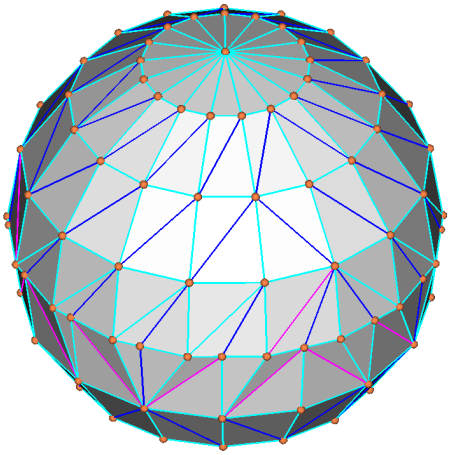
Figure 5-6：An example of SplitEdges(1)



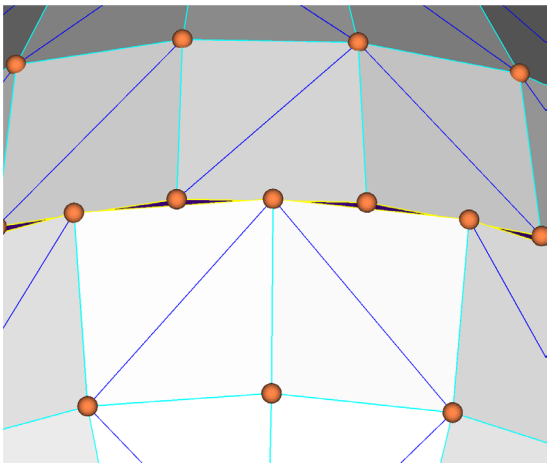
(a) Before processing



(b) After processing (moveVerticesFlag=false)

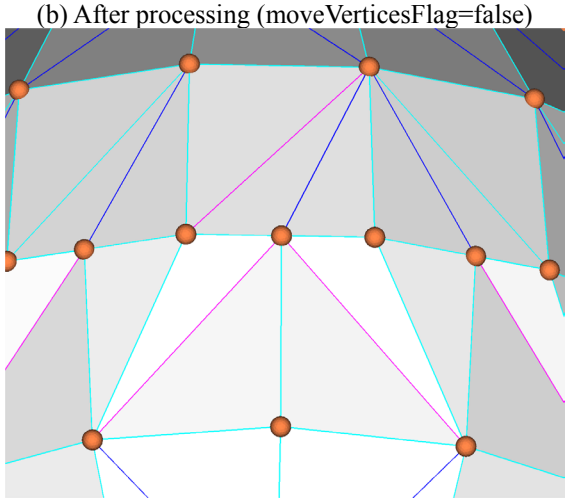


(c) After processing(moveVerticesFlag=true)



(a) Before processing

(b) After processing (moveVerticesFlag=false)



(c) After processing(moveVerticesFlag=true)

Figure 5-7：An example of SplitEdges：An example of SplitEdges(2)

5.4 Piece operations

5.4.1 RemoveClosedPieces/RemoveUnclosedPieces

This method removes closed/unclosed pieces.

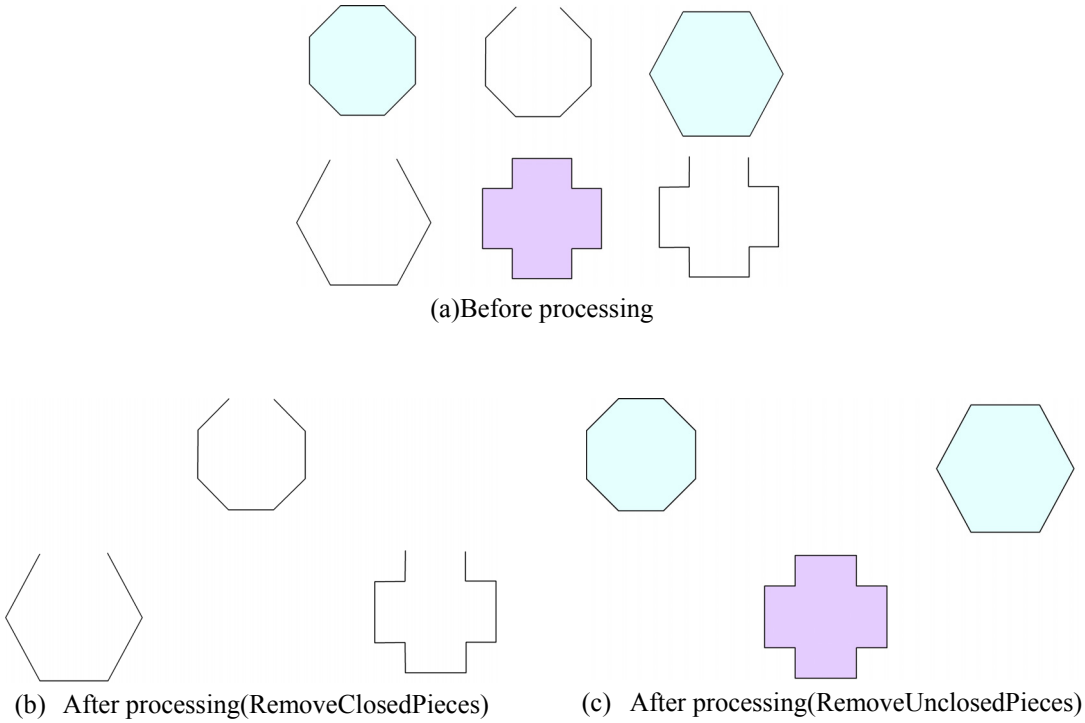
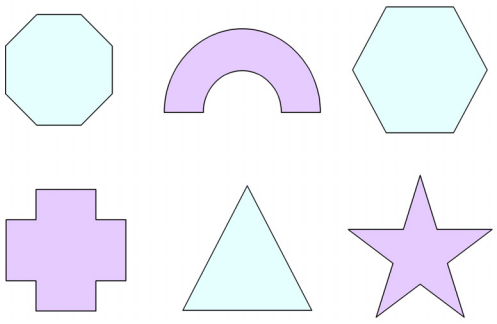


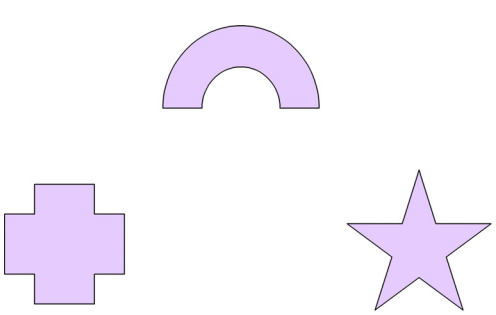
Figure 5-8: Examples of RemoveClosedPieces/RemoveUnclosedPieces

RemoveConvexPieces/RemoveNonConvexPieces

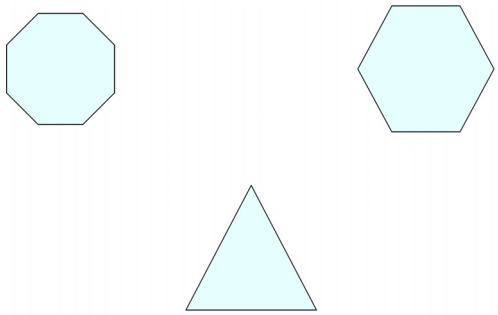
This method removes convex/non-convex pieces.



(a)Before processing.



(b)After processing(RemoveConvexPices)

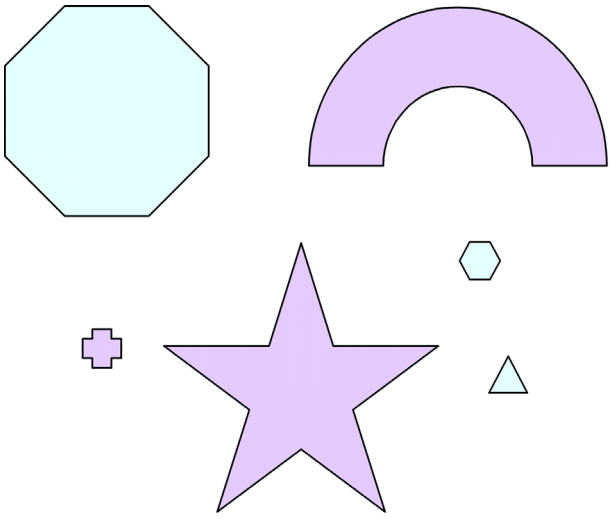


(c) After processing(RemoveNonConvexPices)

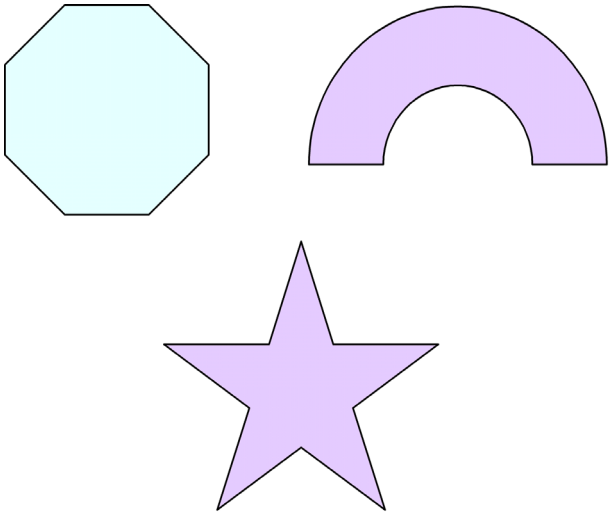
Figure 5-9: Examples of RemoveConvexPieces/RemoveNonConvexPieces

5.4.2 RemoveSmallVolumePieces

This method removes the pieces the volume of which is under tolerance.



(a) Before processing



(b) After processing

Figure 5-10: An example of RemoveSmallVolumePieces

5.4.3 CloseHoles

This method closes holes of found in the original model. Holes are automatically triangulated such that the area of triangles that cover the hole is minimized, without adding any vertices.

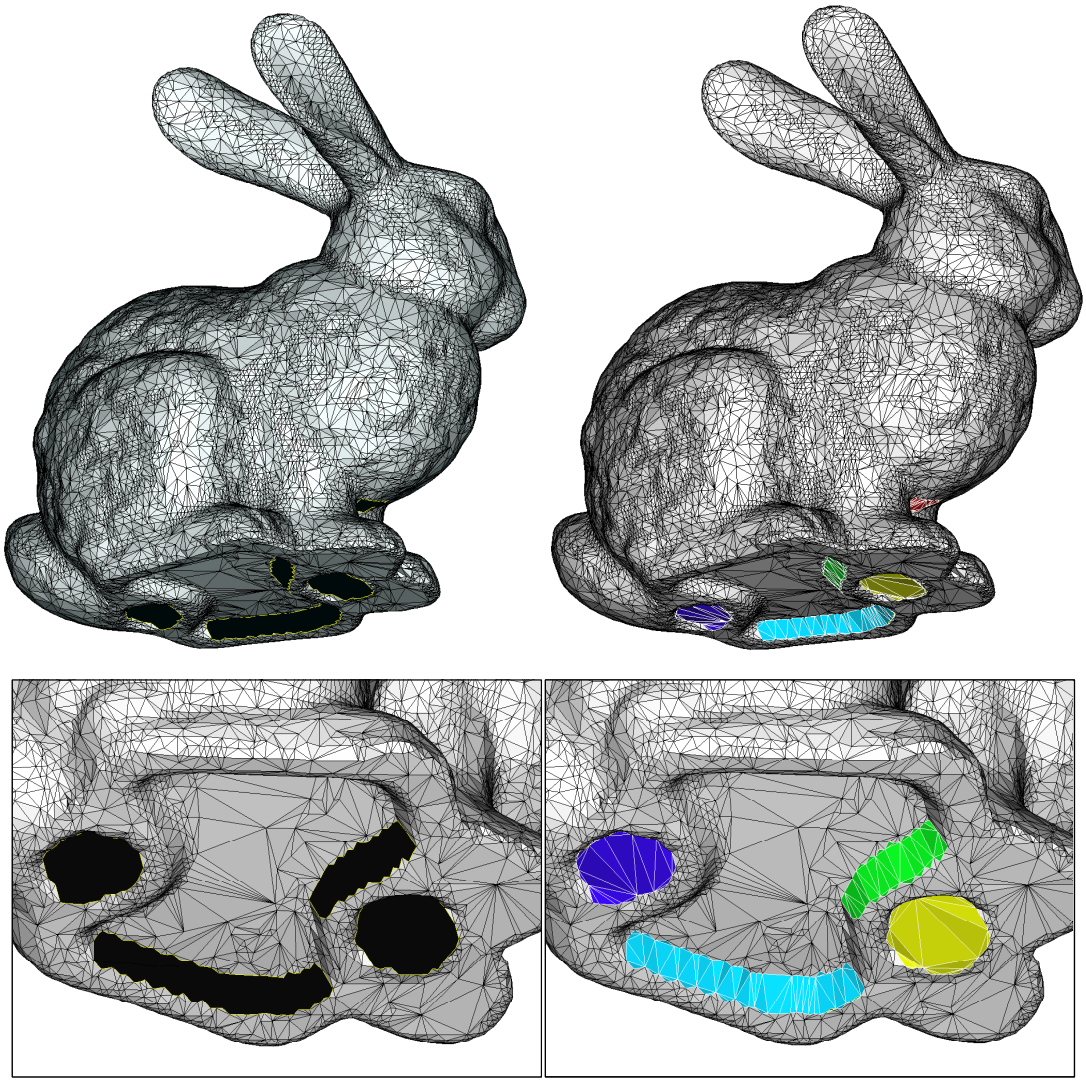
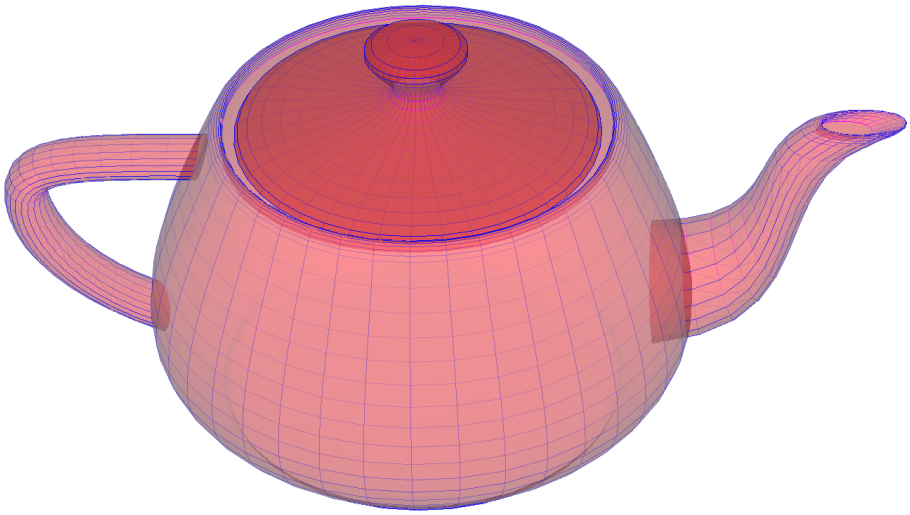


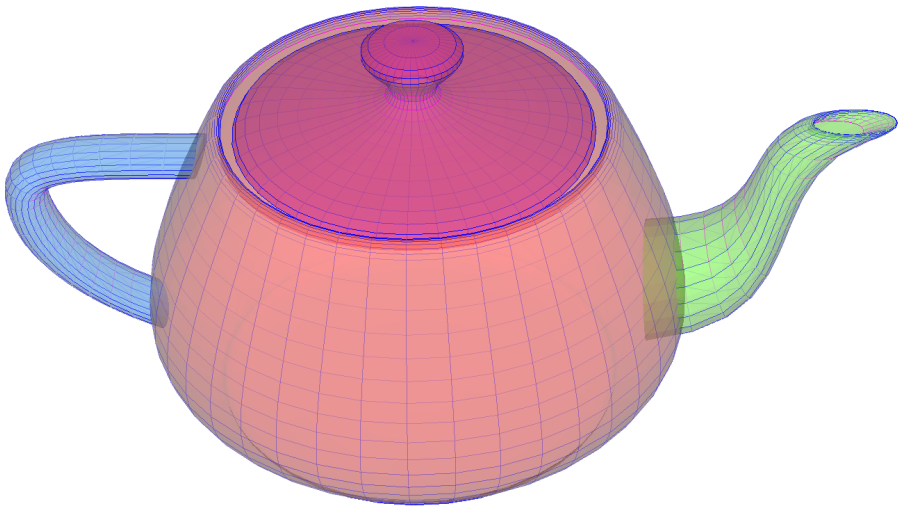
Figure 5-11: An example of CloseHoles

5.4.4 DecomposeIntoSingleBoundaryPieces

This method decomposes the object model into the pieces, each of which has only a single boundary.



(a) Before processing

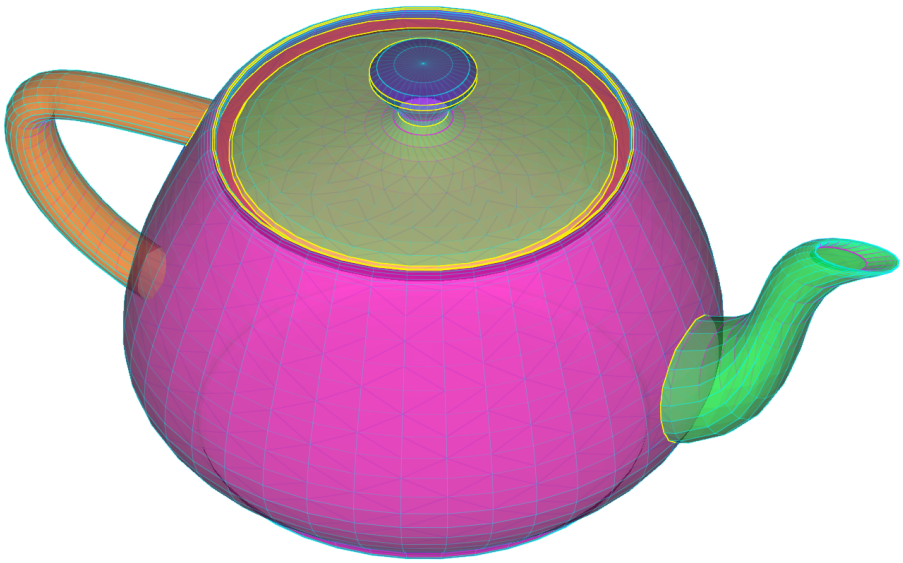


(b) After processing

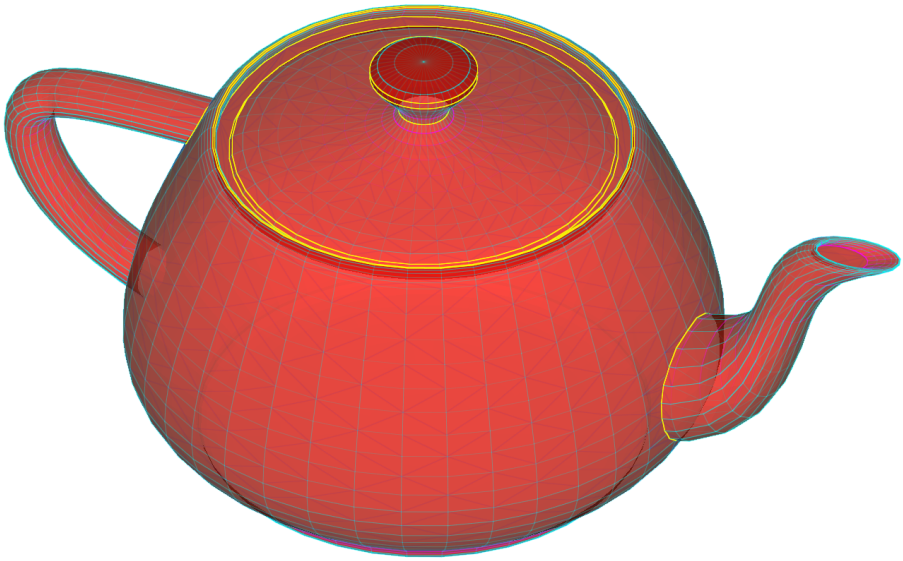
Figure 5-12: An example of DecomposeIntoSingleBoundaryPieces(1)

5.4.5 MergePieces

This method merges mulitple pieces within the object into a single piece.



(a)Before processing



(b) After processing

Figure 5-13: An example of MergePieces

5.5 Triangulation

5.5.1 ChangeTriangulationPattern

This method changes the triangulation pattern of the object model, which means the division of the model's polygons into a set of triangles. Only triangles connected by flat edges are changed, thus the shape always remains the same.

There are various ways to divide a polygon into a set of triangles. Therefore, triangulation is performed from the viewpoint to optimize a particular quantity.

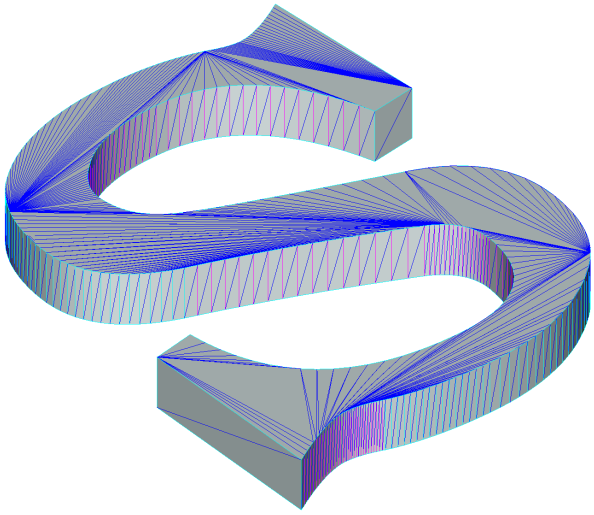


Figure 5-14: Original triangulation pattern

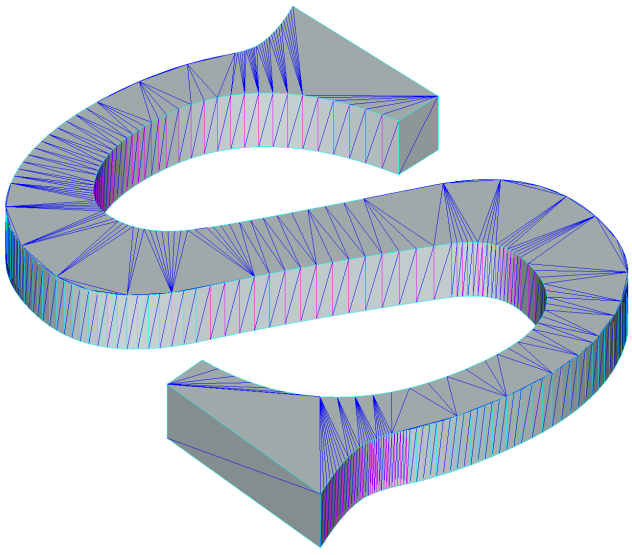


Figure 5-15: Triangulation such that the total edge length is reduced.

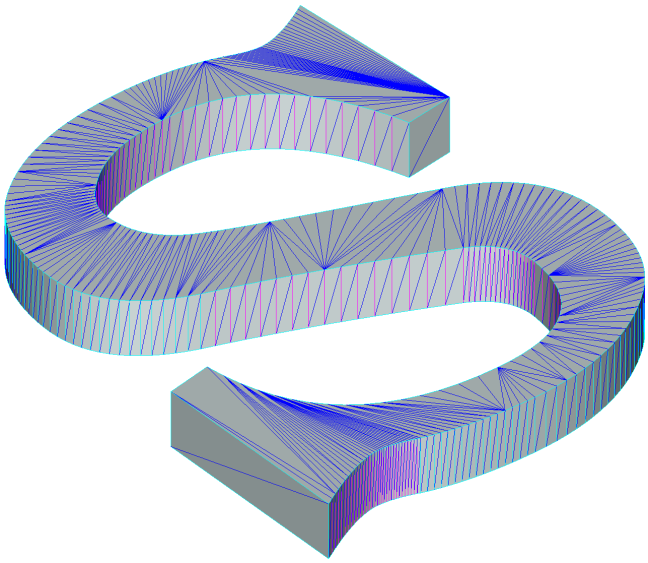


Figure 5-16: Triangulation such that differences of area between triangles are reduced.

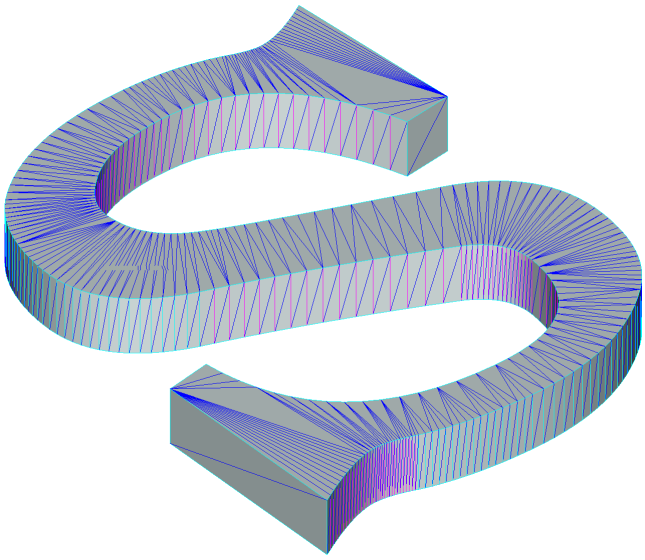


Figure 5-17: Triangulation such that differences of width between triangles are reduced.

6. An example program

This sample program is stored below.

```
examples/SmartPolygonOptimizerTest
```

List 6-1 shows a simple program of SmartPolygonOptimizer API. This program reads the files specified by command line arguments as the input model, constructs SPOObjects from them, prints the results of its diagnosis, modifies them, and prints the diagnosis of the modified model.

List 6-1: SmartPolygonOptimizerTest.cpp

```
#include <stdio.h>
#include <stdlib.h>

#include "spo.h"

// loads object
bool LoadObject(SPOObject&object,const char*file)
{
    FILE*fp;
    fp=fopen(file,"r");
    if(fp==NULL){
          printf("Error: Failed to load file '%s'¥n",file);
          return false;
    }
    int vertexSize=0;
    int faceSize=0;
    // count vertices and faces
    for(;;){
          char s[2000];
          int c;
          c=getc(fp);if(c==EOF)break;ungetc(c,fp);
          fgets(s,2000,fp);
          double v[3];
          int f[3];
          if(sscanf(s,"v %lg %lg %lg",v+0,v+1,v+2)==3)vertexSize++;
          if(sscanf(s,"f %d %d %d",f+0,f+1,f+2)==3)faceSize++;
    }
    rewind(fp);
    double*vertices=new double[vertexSize*3];
    int*faces=new int[faceSize*3];
    int i=0,j=0;
    // store vertices and faces
    for(;;){
          char s[2000];
          int c;
          c=getc(fp);if(c==EOF)break;ungetc(c,fp);
          fgets(s,2000,fp);
          if(sscanf(s,"v %lg %lg %lg",
                    vertices+3*i+0,
                    vertices+3*i+1,
                    vertices+3*i+2)==3){
             i++;
          }
          int f[3];
          if(sscanf(s,"f %d %d %d",f+0,f+1,f+2)==3){
             faces[3*j+0]=f[0]-1;
             faces[3*j+1]=f[1]-1;
             faces[3*j+2]=f[2]-1;
             j++;
          }
```

```
    }
    fclose(fp);
    // addTriangles
    object.AddTriangles(vertices,vertexSize,
                        faces,faceSize);
    delete vertices;
    delete faces;
    return true;
}

// print the result of diagnosis about SPOObject
void Print(SPOObject&object)
{
    int i;
    printf("Closed:%s¥n",object.IsClosed()?"true":"false");
    printf("Convex:%s¥n",object.IsConvex(1e-3)?"true":"false");
    printf("Single piece:%s¥n",object.IsSingleBoundary()?"true":"false");
    printf("PicesSize=%d¥n",object.GetPieceCount());
    for(i=0;i<object.GetPieceCount();i++){
          const SPOPiece*p=object.GetPiece(i);
          printf(" #%d¥n",i);
          printf("  vertex size: %d¥n",p->vertexSize);
          printf("  triangle size: %d¥n",p->triangleSize);
          printf("  Closed:%s¥n",object.IsClosed(i)?"true":"false");
          printf("  Convex:%s¥n",object.IsConvex(i,1e-3)?"true":"false");
          printf("  Single piece:%s¥n",object.IsSingleBoundary(i)?"true":"false");
          printf("  Branched  edge:%d¥n",object.GetEdgeCount(SPO_EDGE_TYPE_BRANCHED,i));
          printf("  Duplicate edge:%d¥n",object.GetEdgeCount(SPO_EDGE_TYPE_DUPLICATE,i));
          printf("  Unlinked  edge:%d¥n",object.GetEdgeCount(SPO_EDGE_TYPE_UNLINKED,i));
          printf("  Folding   edge:%d¥n",object.GetEdgeCount(SPO_EDGE_TYPE_FOLDING,i,1e-3));
    }
    printf("¥n");
}

int main(int argc, char* argv[])
{
    int i;
    for(i=1;i<argc;i++){
       SPOObject object;
          LoadObject(object,argv[i]);
          printf("***Imformation of raw model data***¥n");
          Print(object);
          // Insert methods to modify object
          object.ConnectVertices(1e-4);
          object.SplitEdges(2,20);
          object.CloseHoles();
          object.RemoveRedundantVertices(0.01,20);
          object.RemoveThinTriangles(0.05,20);
          object.ChangeTriangulationPattern(SPO_TRIANGULATION_TYPE_REDUCE_WIDTH_DIFFERENCE,0.01,20);
          object.RemoveRedundantVertices(0.01,20);
          object.DecomposeIntoSingleBoundaryPieces();
          printf("***Imformation of modified model data***¥n");
          Print(object);
    }
    system("pause");
    return 0;
}
```