SmartCollision™ SDK
User's manual

Version 2.01

January 12, 2007

3D Inc.

3D Incorporated

3D Incorporated LICENSE AGREEMENT

3D Incorporated

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.

SmartCollision™ SDK

version 2.01

Copyright

Trademarks

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. Introduction

SmartCollisionSDK is a class library which can handle collisions between virtual objects represented by polygonal models. SmartCollisionSDK has mainly two types of collision detection, namely **minimum distance computation** and **penetration depth computation**. If there is no collision between objects, the minimum distance computation is performed. If there are collisions between objects, the penetration depth computation is performed.

One of the main features of SmartCollisionSDK is high performance collision detection, and the ability to calculate penetration depth between non-convex polyhedra. This ability provides for realistic collision resolution between objects in a Virtual Reality system, such as a digital mockup, haptics applications, or other advanced simulations. For haptics applications, SmartCollisionSDK enables 6 DOF (Degrees-of-Freedom) for force feedback manipulation. Figure 1-1 shows an example of a haptics application using SmartCollisionSDK.

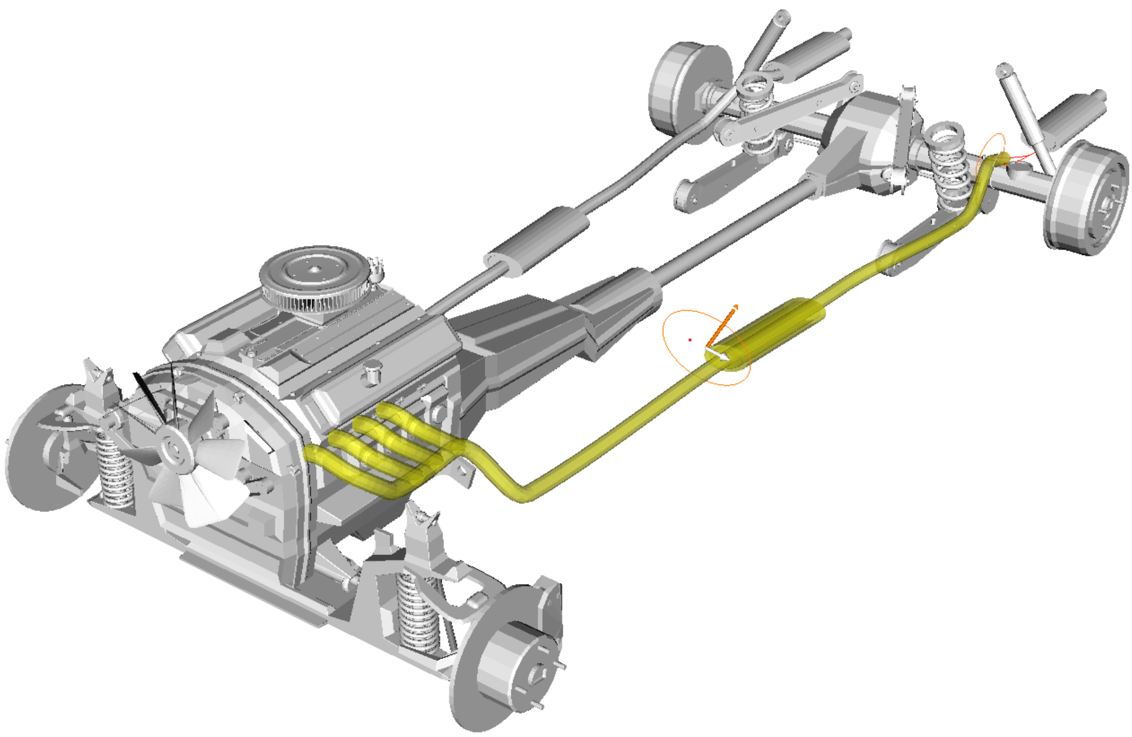


Figure 1-1: An example of a haptics application

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 protect key)

Platforms

Microsoft Windows 2000 or XP

Linux (Fedora Core 5 or later)

Compiler

Microsoft Visual C++ 6.0 or later

gcc (GCC) 4.1.0 or later

Requirements for some of the examples

GLUT 3.7(The OpenGL Utility Toolkit)

http://www.opengl.org/resources/libraries/glut/glut_downloads.html

Optional Requirements for "SmartCollsionTest" example (only for PHANTOM application)

SensAble PHANTOM device and

Open Haptic Toolkit 1.00 or later

2.2 Installation

SmartCollision SDK does not have the installer program.

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

2.3 License Activation

Before run the application that uses SmartCollision SDK, 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 SDK, sc.h must be included in source files of your project and sc.lib must also be linked. At run time, sc.dll is required.

spo is a directory in which the SmartPolygonOptimizerAPI is stored. SmartPolygonOptimizerAPI is an API which helps you to import polygonal data into your project.

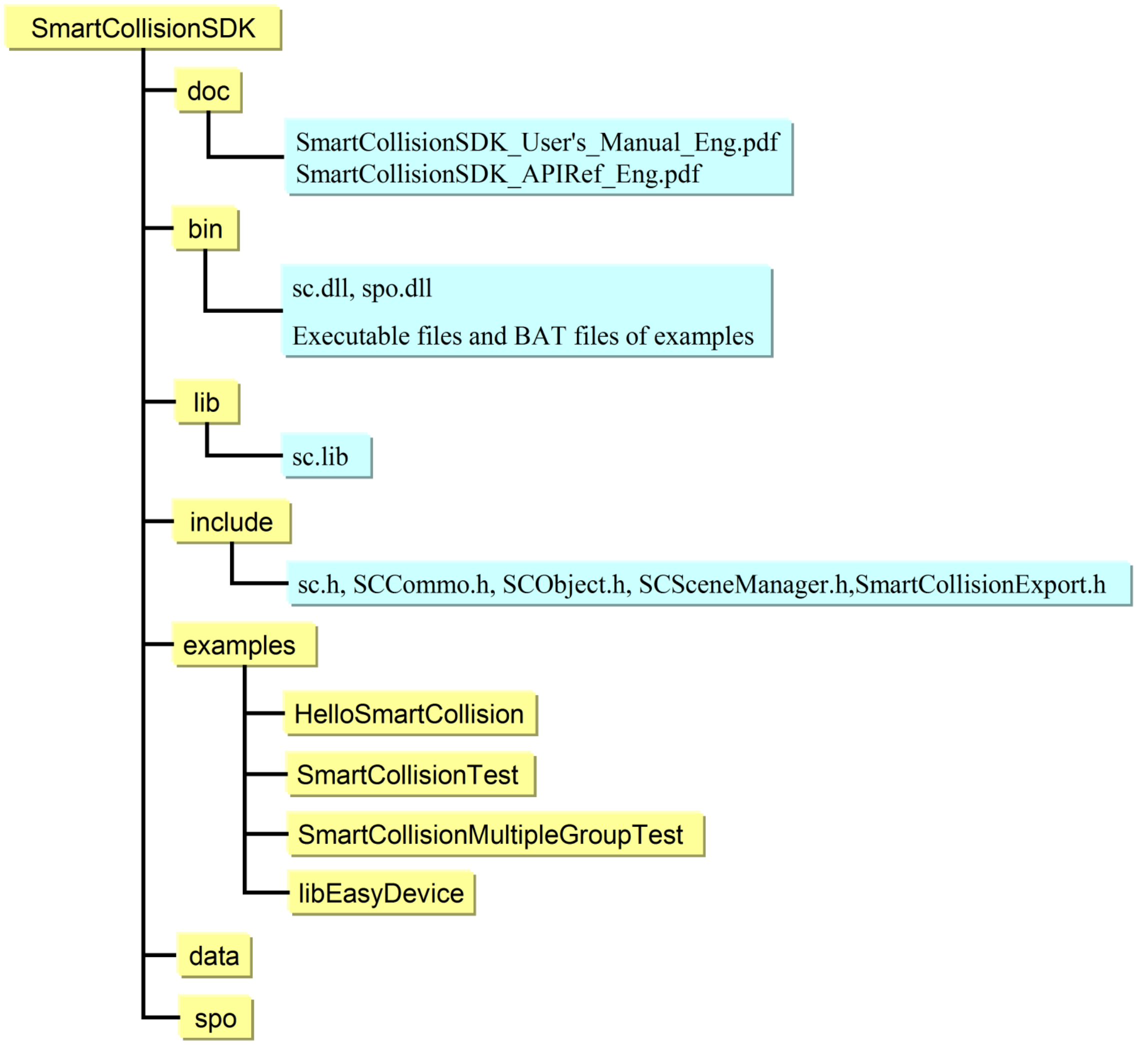


Figure 2-1: Files of SDK

2.4.2 Linux version

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

spo is a directory in which the SmartPolygonOptimizerAPI is stored. SmartPolygonOptimizerAPI is an API which helps you to import polygonal data into your project.

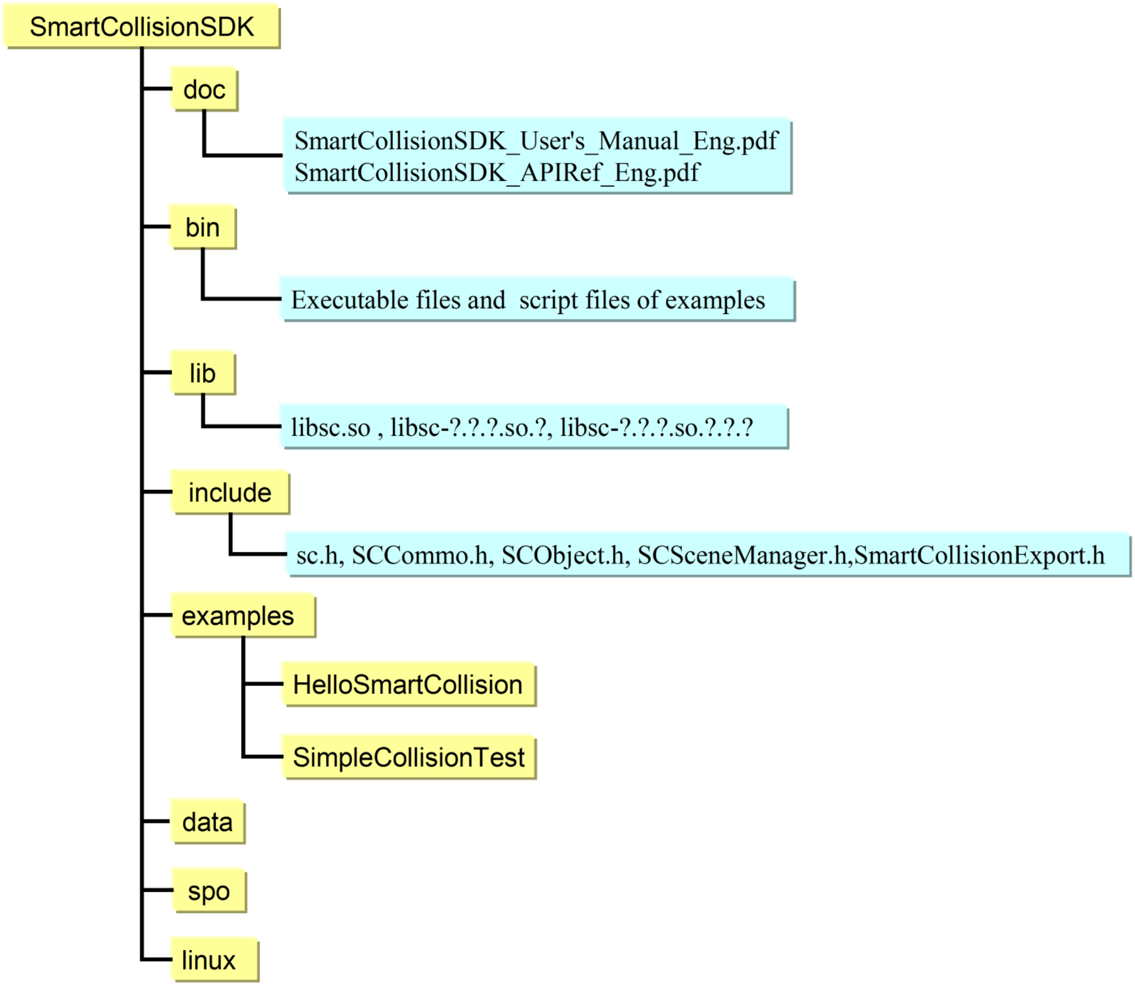


Figure 2-2: Files of SDK for Linux

3. Class interface

SmartCollision SDK is implemented as a C++ class library, and consists of following two classes.

SCObject : object class(SmartCollision Object)

SCSceneManager : scene class (SmartCollision Scene Manager)

SCObject is a class which stores its own geometry and transformation information. Geometry is defined as a polygonal object which consists of a set of triangles.

SCSceneManager is a class which manages objects in the scene. SCObjects are added to SCSceneManager and are managed by unique IDs which are assigned to the object by the application developer.

Every object in the scene belongs to a group, which also have user defined IDs. The SCSceneManager performs collision detection between pairs of groups, which can be controlled with respect to the attributes of groups, and pairs of groups. When there is no contact between a pair of groups, minimum distance computation is performed. When an object in a group has collided with another, and one object has penetrated the bounds of the other, penetration depth computation is performed. Penetration depth computation continues as long as penetration depth is not zero.

4. Basic theory

4.1 Geometric definition of penetration depth

Penetration Depth is a measure of penetration. As Figure 4-1 shows, when there is an intersection between object A and B, the geometric definition of **Penetration Depth Vector** ($\boldsymbol{d}$) is defined as the globally minimum distance vector which cancels the penetration. The penetration depth vector can be written as （ 4-1 ）.

$$\min\{\|\boldsymbol{d}\| \mid interior(A+\boldsymbol{d}) \cap B = \varnothing\} \qquad (4\text{-}1)$$

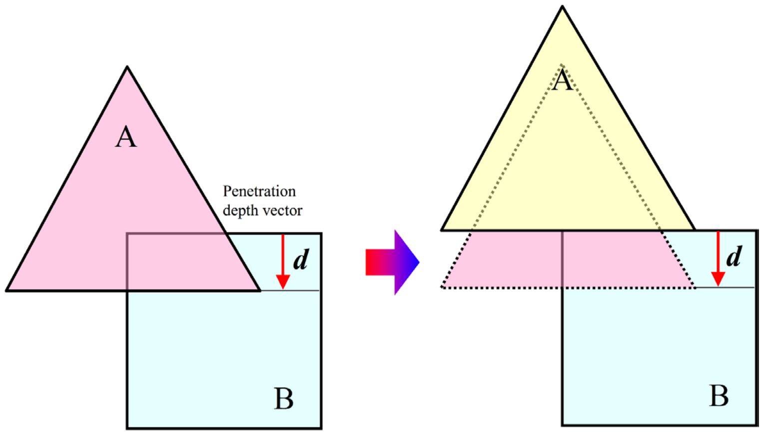


Figure 4-1: Penetration depth

However, the penetration depth vector defined by （ 4-1 ） is not a continuous function of the (continuous) motion of A. For example, when A moves continuously as shown in Figure 4-2, the penetration depth vector changes from $\boldsymbol{d}$ to $\boldsymbol{d}'$. The size of penetration depth vector is continuous but, the direction changes discontinuously.

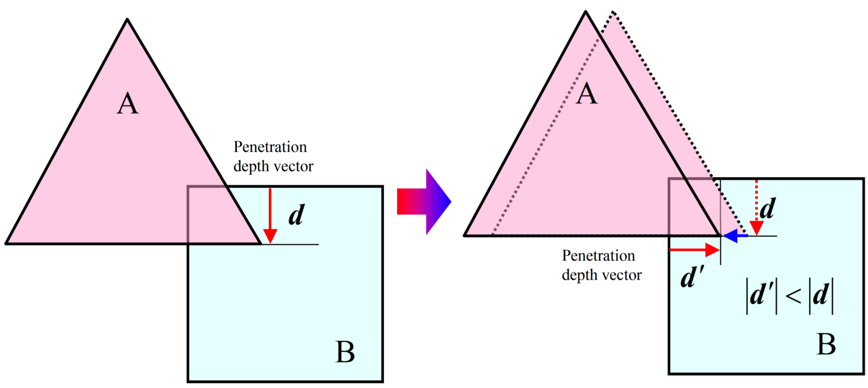


Figure 4-2: Discontinuous change of penetration depth vector

4.2 Definition of penetration depth by CSO

Penetration depth can also be defined by **CSO (Configuration Space Obstacle)**. The CSO is defined as the Minkowski difference between objects, which can be written as （ 4-2 ）.

$$A \oplus -B = \{\boldsymbol{p} + \boldsymbol{q} \mid \boldsymbol{p} \in A, \boldsymbol{q} \in -B\} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (4\text{-}2)$$

Figure 4-3 shows the CSO of A and B and penetration depth defined by the CSO, when object A and B are penetrating. Figure 4-4 shows how to make CSO.

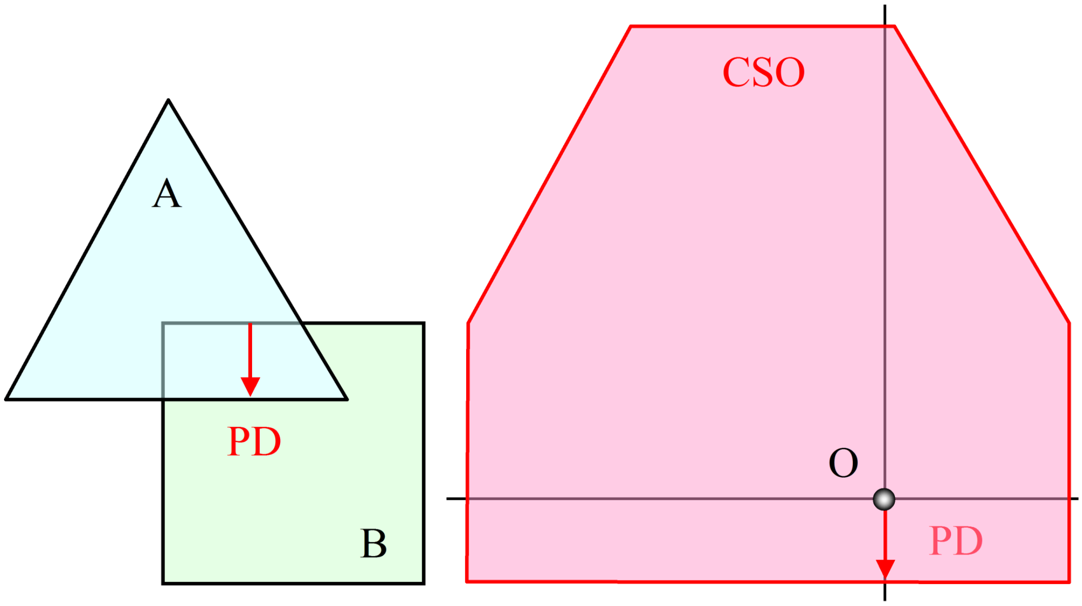


Figure 4-3: CSO between A and B

When the origin of the coordinate system for A and B is on the boundary of the CSO, A and B are just touching. When the origin is inside the CSO, the boundaries for A and B are intersecting, and the penetration depth vector is defined as the minimum distance vector from the origin to the boundary of the CSO.

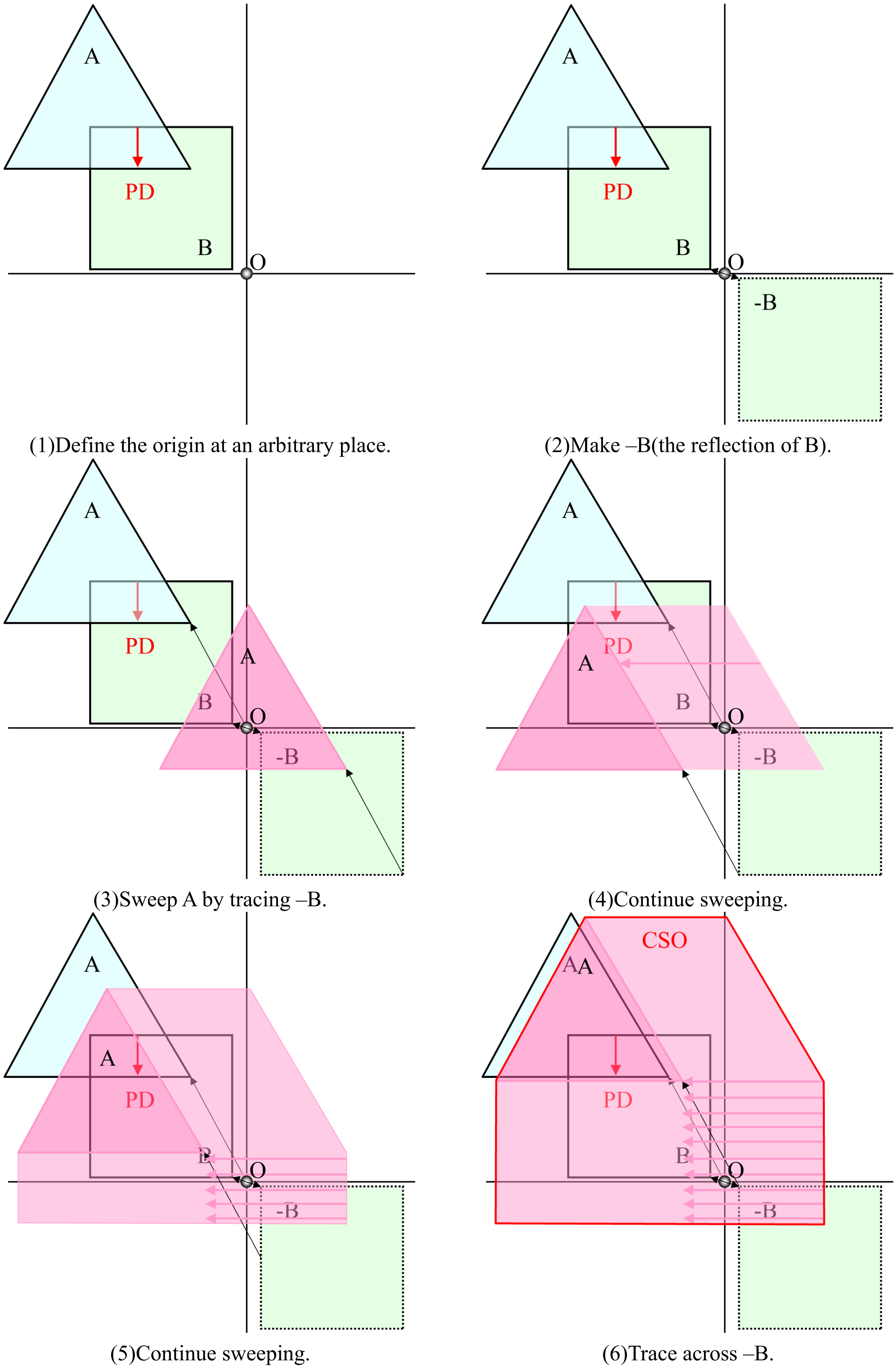


Figure 4-4: How to make CSO

4.3 The local minimum penetration depth vectors

The penetration depth vector is usually defined as the global minimum of all local minimum translational vectors from the origin. In order to find the global minimum, it is necessary to find all the local minima. The local minimum translational vectors are all minimum vectors within the CSO that can resolve the intersection between A and B. Therefore they are also the local minimum penetration depth vectors.

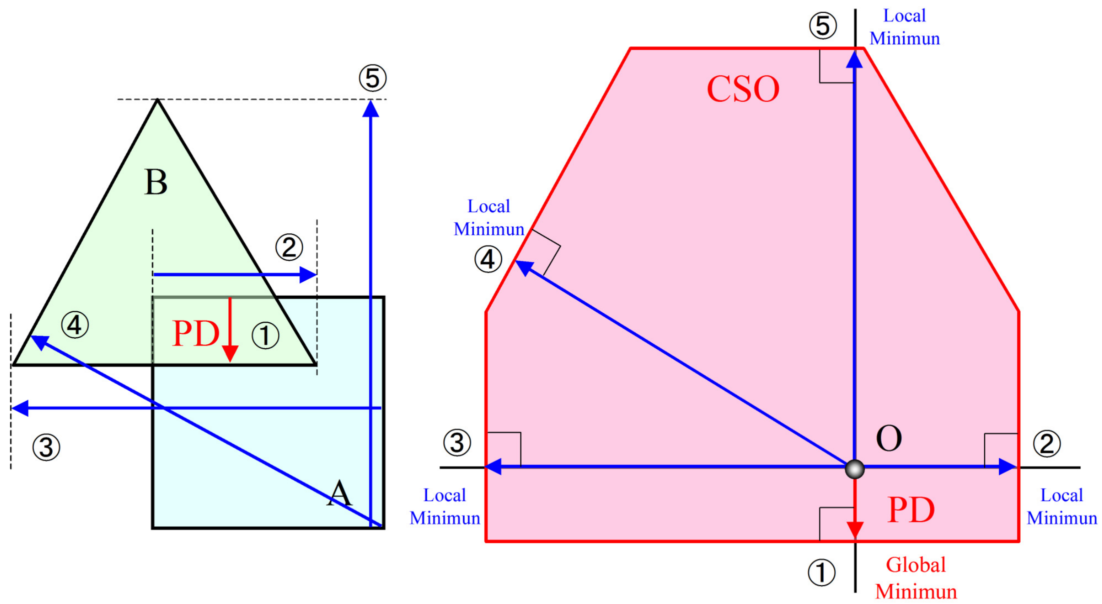


Figure 4-5: Local minima of PD

4.4 Dynamic penetration depth vector

If the origin is fixed, then the CSO moves as A and B move. When A and B start intersecting, the origin enters into the CSO at a point. This point is defined as the contact point of the CSO. While A and B move continuously, the contact point is assumed to move in the direction that decreases the distance from the origin. The contact point also must move continuously. In this situation, the **Dynamic Penetration Depth Vector** is defined as the vector from the origin to the contact point.

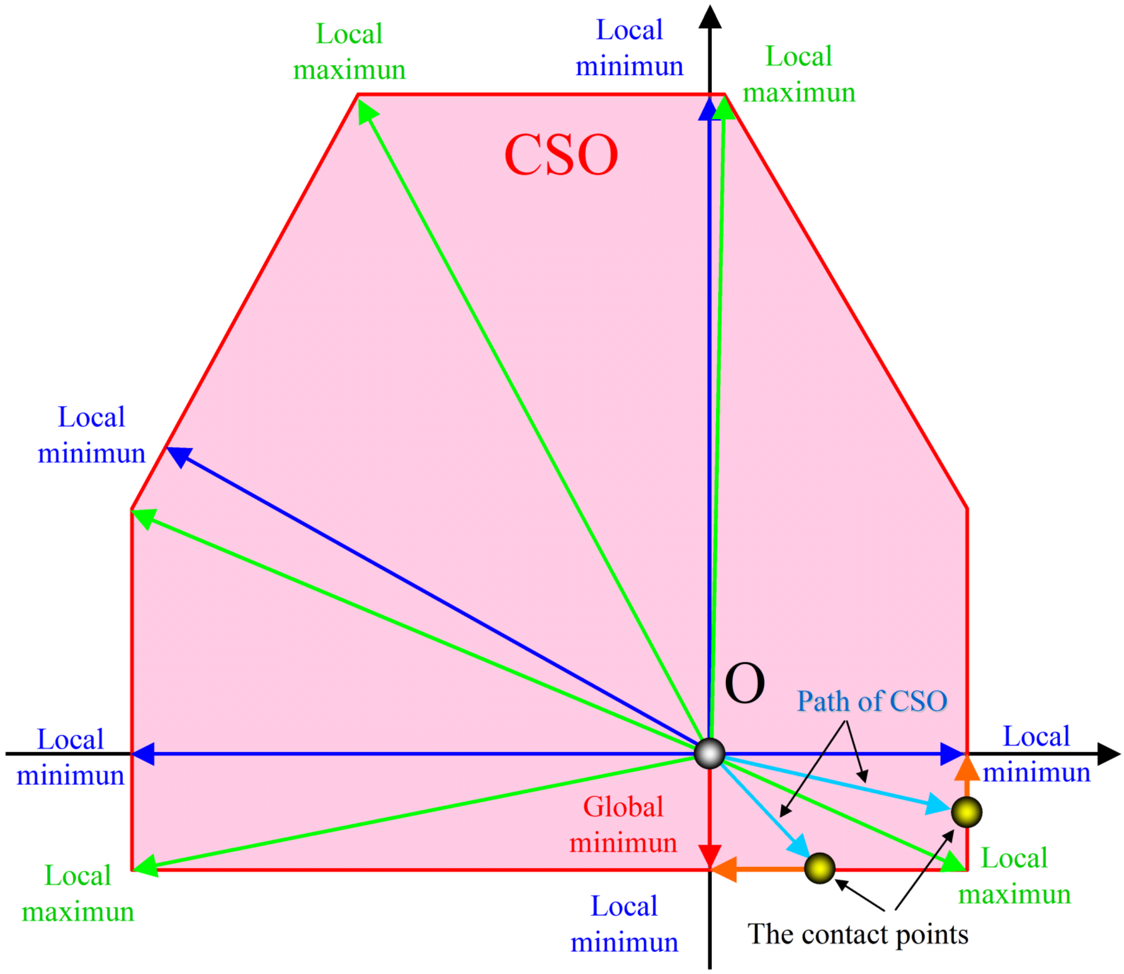


Figure 4-6: Dynamic penetration depth vector from the point of view of CSO

In Figure 4-7, $\boldsymbol{d}$ is the geometric representation of the penetration depth vector (the global minimum), and $\boldsymbol{D}$ is the dynamic penetration depth vector. In (a), $\boldsymbol{d}$ is equal to $\boldsymbol{D}$, but in (b) $\boldsymbol{d}$ is not. In this manual, the dynamic penetration depth vector is simply called penetration depth vector.

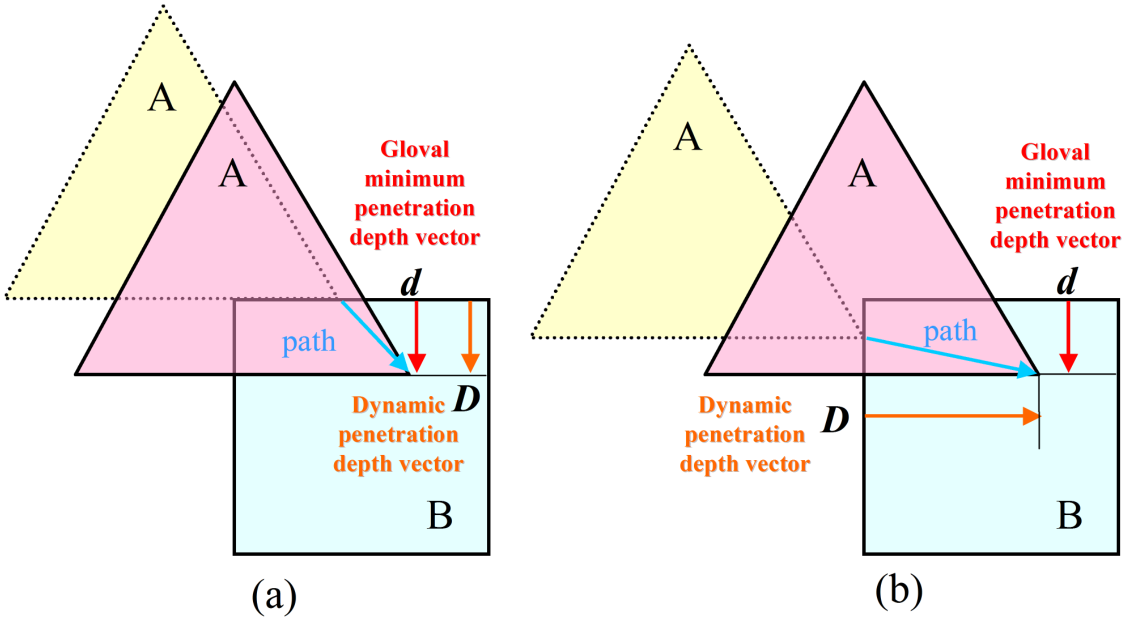


Figure 4-7: Dynamic penetration depth vector

4.5 Partial penetration depth and total penetration depth

When the object A and B intersect as shown in Figure 4-8 (a), the penetration depth vectors about each convex piece can be obtained as shown in Figure 4-8 (b). The **Partial Penetration Depth Vector** is defined as the penetration depth vector about each convex piece. However, simply moving the object A according to the partial penetration depth vector is insufficient to resolve the object intersection. On the other hand, moving the penetrating object according to the **Total Penetration Depth Vector**, which is obtained from CSO as shown in Figure 4-9, can resolve the object intersection. In the manual, the total penetration depth vector is simply called penetration depth.

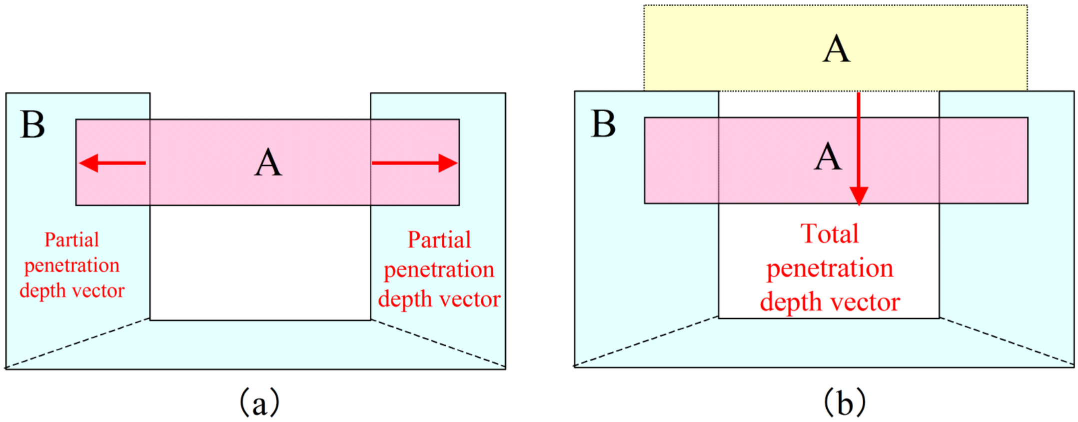


Figure 4-8: Partial penetration vector and total penetration vector

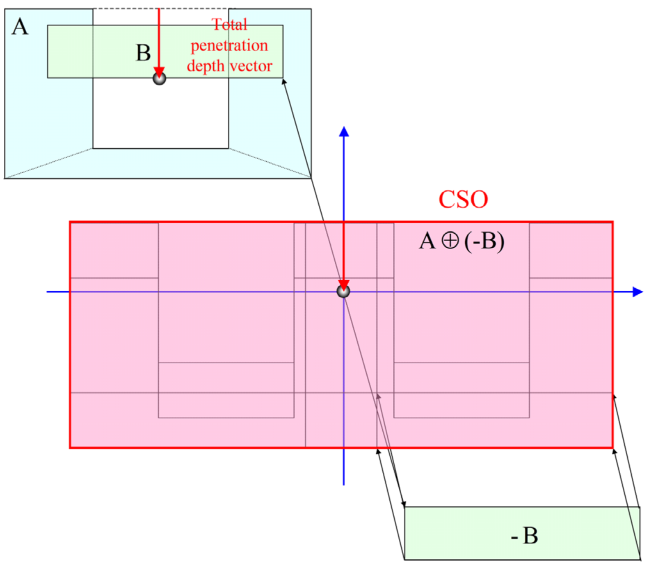


Figure 4-9: Total penetration depth obtained from CSO

4.6 Multiple contacts

Figure 4-10 (a) shows object B intersecting with object A with penetration depth $\boldsymbol{D}$. If object B moves according to the vector $-\boldsymbol{D}$, the intersection is resolved and object B is in contact with object A at two points. Figure 4-10 (b) shows the CSO of A and B while they are in a penetration state. In the CSO, multiple contacts points are projected into a single point. In order to resolve the object intersection between A and B, it is sufficient to calculate penetration depth vector, but not necessarly to list all contact points.

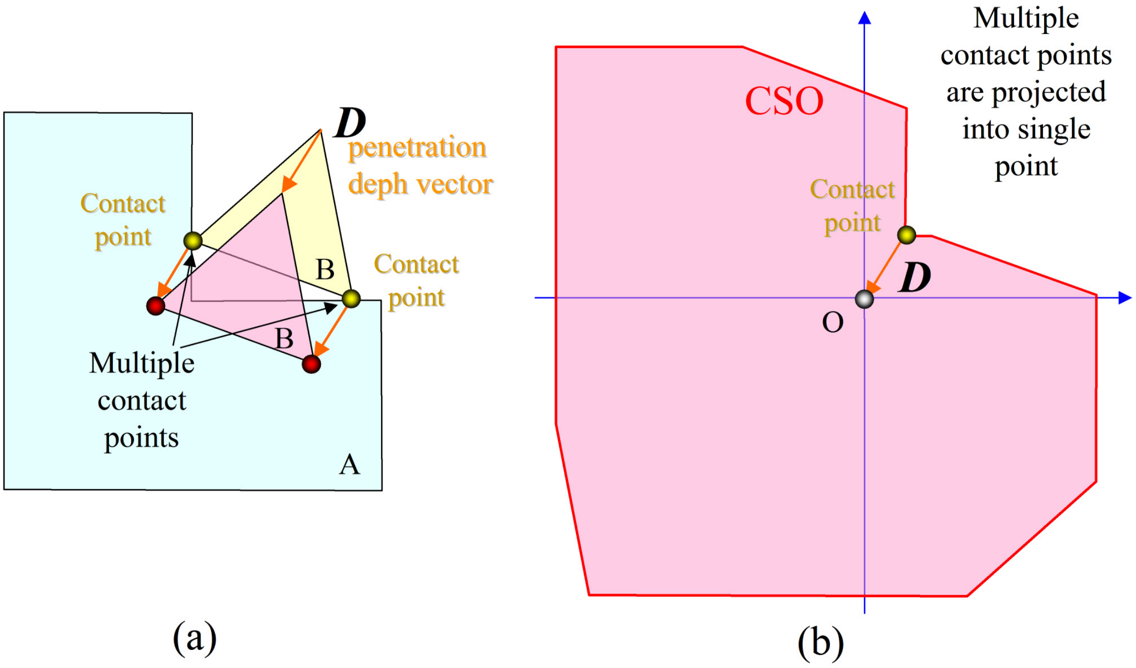


Figure 4-10: Multiple contacts

4.7 Generalization of penetration depth

So far, the motions of objects are limited to translational motion. However, when objects rotate, it is not always possible to cancel the penetration between objects only by applying a reverse translation. Figure 4-11 shows an example in which the boundary of the CSO disappears, and the penetration cannot be resolved directly with only a translation.

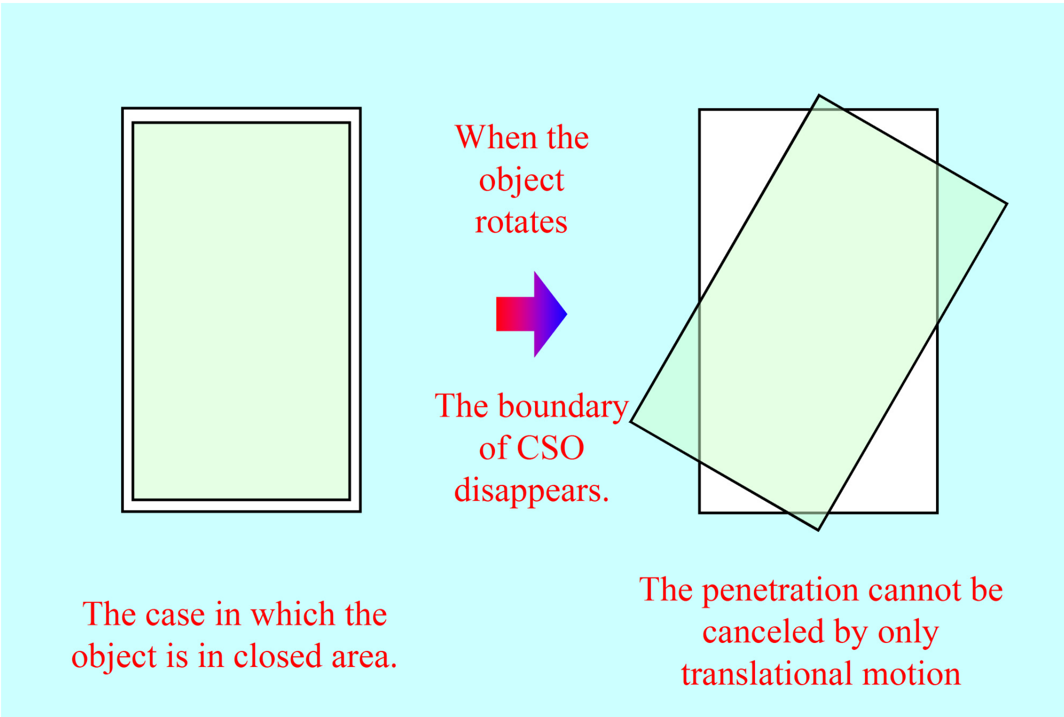


Figure 4-11: An example in which boundary of CSO disappears.

To resolve a penetration which is caused by a rotational motion, it is natural to introduce a rotation factor to the penetration depth vector, and split the penetration depth vector into translational and rotational parts; namely **TPDV** (**T**ranslational **P**enetration **D**epth **V**ector) and **RPDV** (**R**otational **P**enetration **D**epth **V**ector).

Figure 4-13 shows the case in which introduction of TPDV D_T and RPDV D_R enables to cancel the penetration between objects with less movement. Here, $D_R/|D_R|$ specifies the axis of rotation and , $|D_R|$ means the amount of rotation.

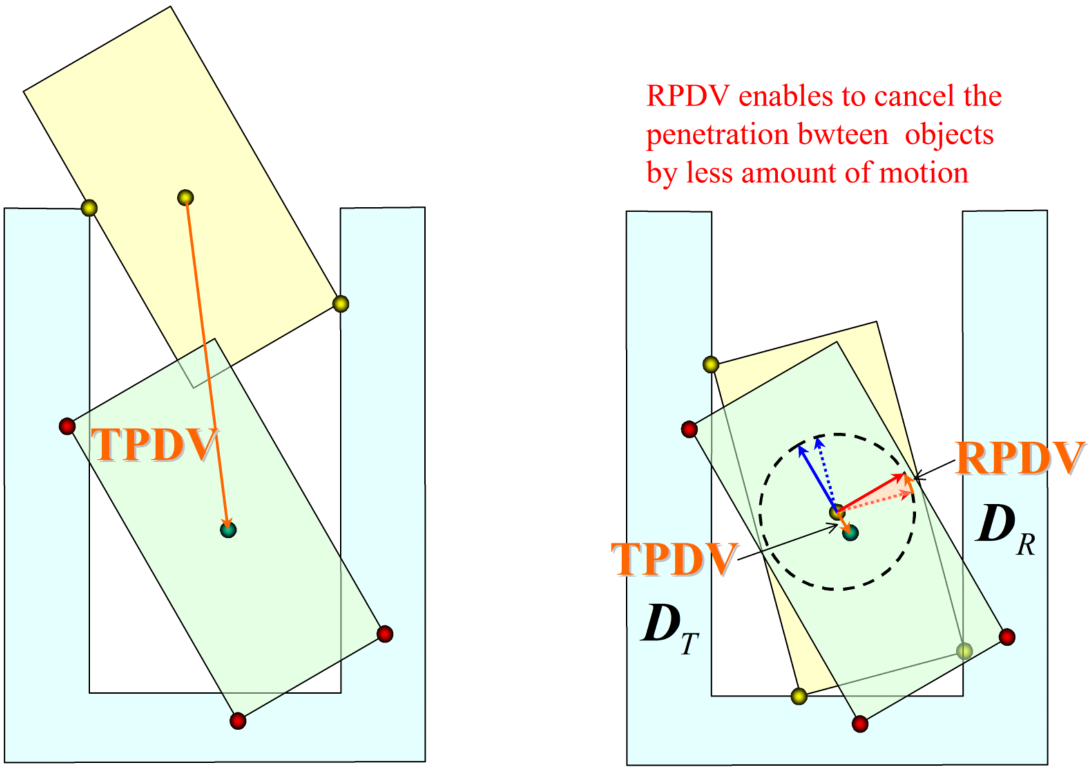


Figure 4-12: Introduction of TPDV and RPDV

4.8 Arbitrariness about TPDV and RPDV

To extend the notion of penetration depth to rotational motion, RPDV was introduced. However, the combination of TPDV and RPDV cannot be determined uniquely. TPDV depends on RPDV and vice versa. Figure 4-13 (a) shows the case in which RPDV is zero, (b) shows the case in which TPDV is zero. (c) shows the case in which RPDV is minimized, but TPDV is not zero because of the other constraint.

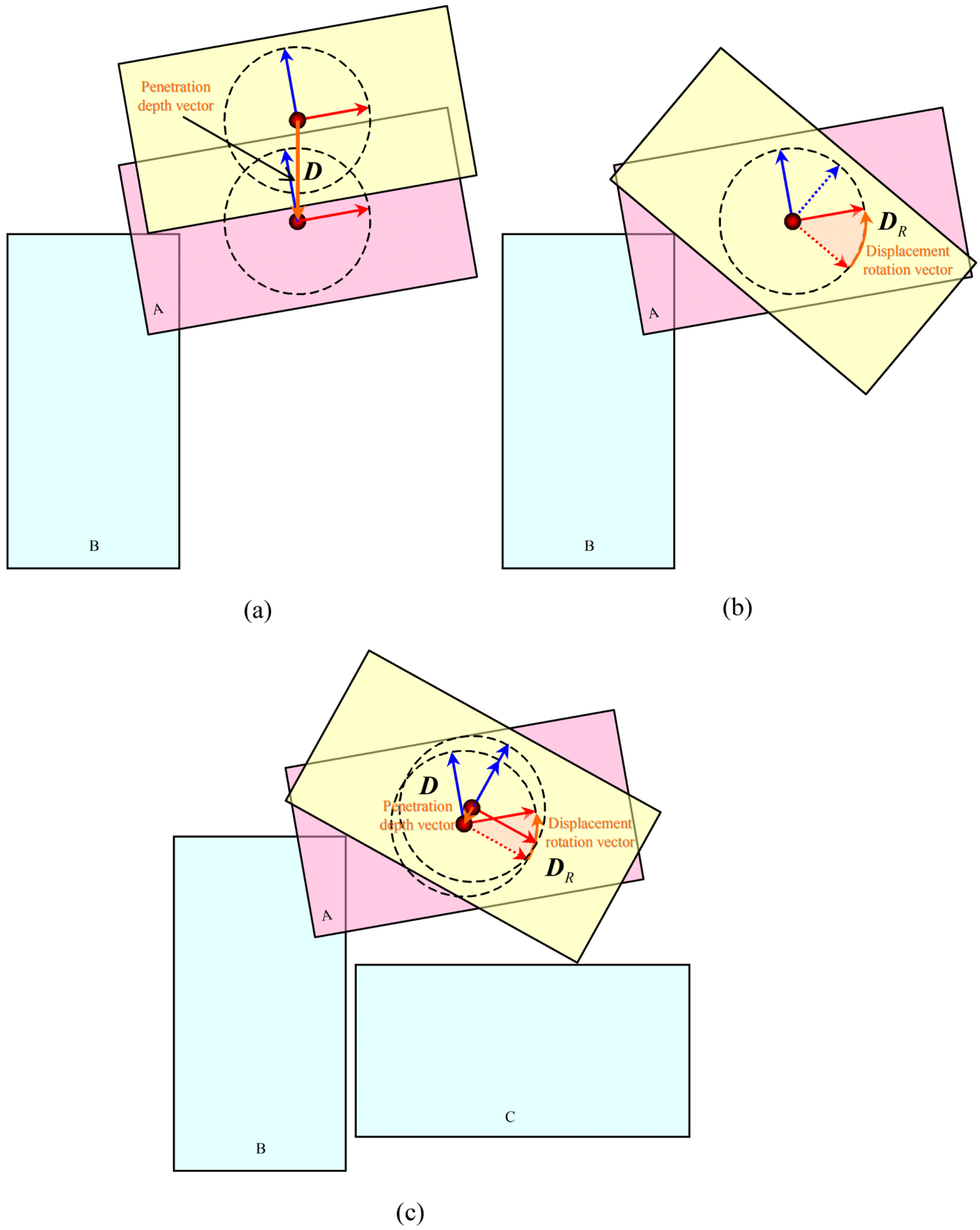


Figure 4-13: Combinations of TPDV and RPDV

4.9 Measurement of object penetration

As mentioned before, the combination of the TPDV and RPDV cannot be determined uniquely for a given penetrating state. This arises from the fact that translation and rotation cannot be compared directly. Therefore, it is necessary to introduce a method to measure object penetration when A and B intersect using some combination of TPDV and RPDV.

To avoid this problem, let us consider the situation in which the object is penetrating with a TPDV of $\boldsymbol{D}_T$ and a RPDV of $\boldsymbol{D}_R$. The force $\boldsymbol{F}$ and the torque $\boldsymbol{\tau}$ acting on the object are given by (4-3), (4-4).

$$\boldsymbol{F} = -k_T \boldsymbol{D}_T \quad \text{(4-3)}$$

$$\boldsymbol{\tau} = -k_R \boldsymbol{D}_R \quad \text{(4-4)}$$

Here, $k_T[\mathrm{N/m}], k_R[\mathrm{N \cdot m}]$ are the coefficients of stiffness for the force and torque equations.

In this case, the potential energy of the object $P(\boldsymbol{D}_T, \boldsymbol{D}_R)$ is given by (4-5).

$$P(\boldsymbol{D}_T, \boldsymbol{D}_R) = \frac{1}{2} k_T \left|\boldsymbol{D}_T\right|^2 + \frac{1}{2} k_R \left|\boldsymbol{D}_R\right|^2 \quad \text{(4-5)}$$

$P(\boldsymbol{D}_T, \boldsymbol{D}_R)$ can be used as the value to measure object penetration. A desirable combination of $\boldsymbol{D}_T$ and $\boldsymbol{D}_R$ can be determined such that $P(\boldsymbol{D}_T, \boldsymbol{D}_R)$ has a locally minimum value.

4.10 Bi-directional Penetration Depth Computation

So far, only one object (or a group of objects) is moving and the other object (or a group of objects) is resting, and penetration depth is calculated with respect to the moving object against the resting object. In more general situations such as rigid body dynamics, both objects (or groups of objects) are moving.

However, the combination of TPDV and RPDV with respect to the one object cannot be calculated from the other by simply reversing direction of each vector. This arises from the fact that the RPDV depends on the center of rotation and center of rotation might be different from each other.

One solution for this problem is **Bi-directional Penetration Depth Computation**. In this solution, there are two combinations of TPDV and RPDV for one penetration between the two objects (or two groups of objects).

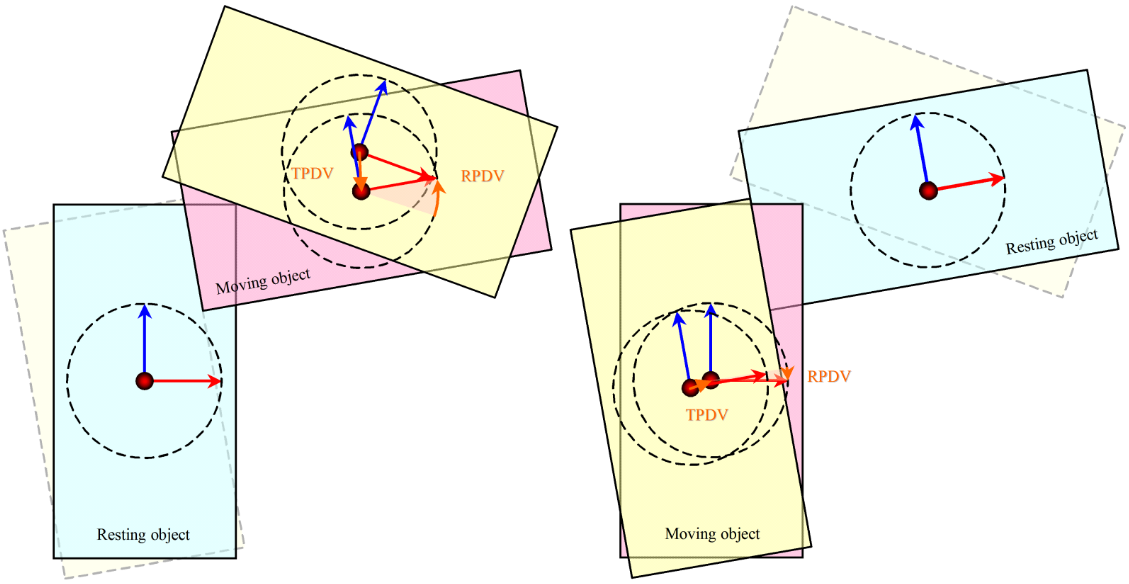


Figure 4-14: Bi-directional penetration depth computation

5. Input geometry

There are two types of input geometry for SmartCollisionSDK. **Triangle soup** and **closed polyhedra**. Figure 5-1 shows the category of input geometry for SmartCollisionSDK. Triangle soup is more general polygonal model. Triangle soup includes closed polyhedra and closed polyhedra include convex polyhedra.

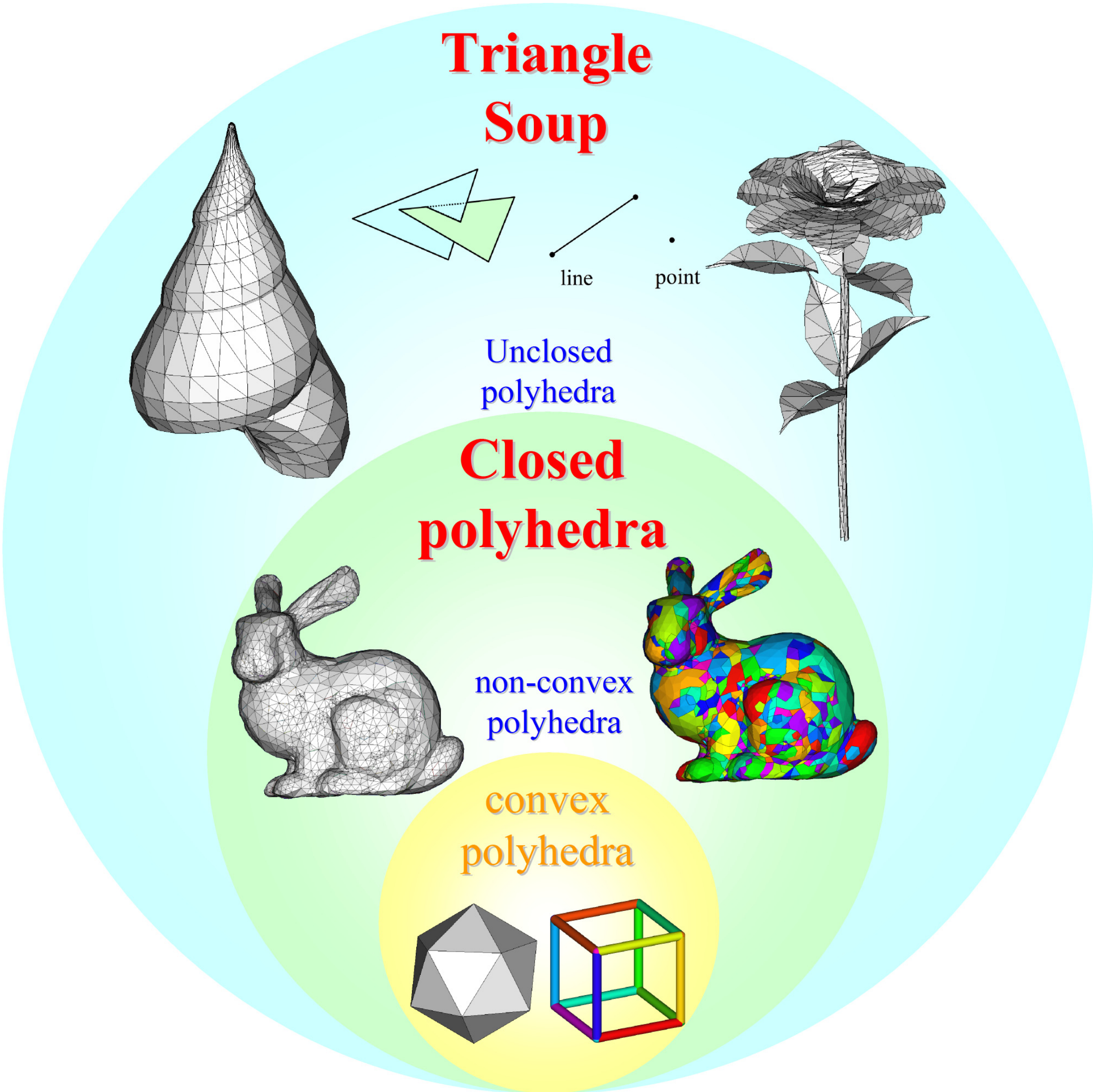


Figure 5-1: Input geometry for SmartCollisionSDK

5.1 Triangle soup

Triangle soup is arbitrary set of triangles. Triangle soup has few limitations and preprocessing is fast, but performance of collision detection is relatively slow. As shown in Figure 5-2, triangle soup allows intersecting triangles or triangles degenerating into line or point. Figure 5-3 shows an example of triangle soup.

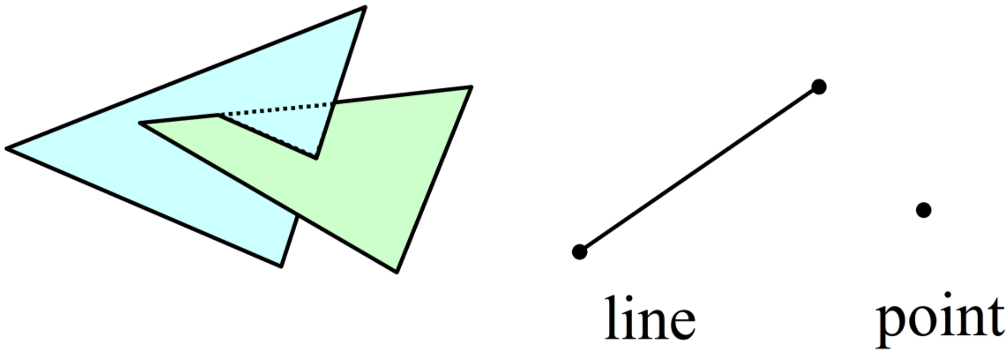


Figure 5-2: Possible case of triangle soup

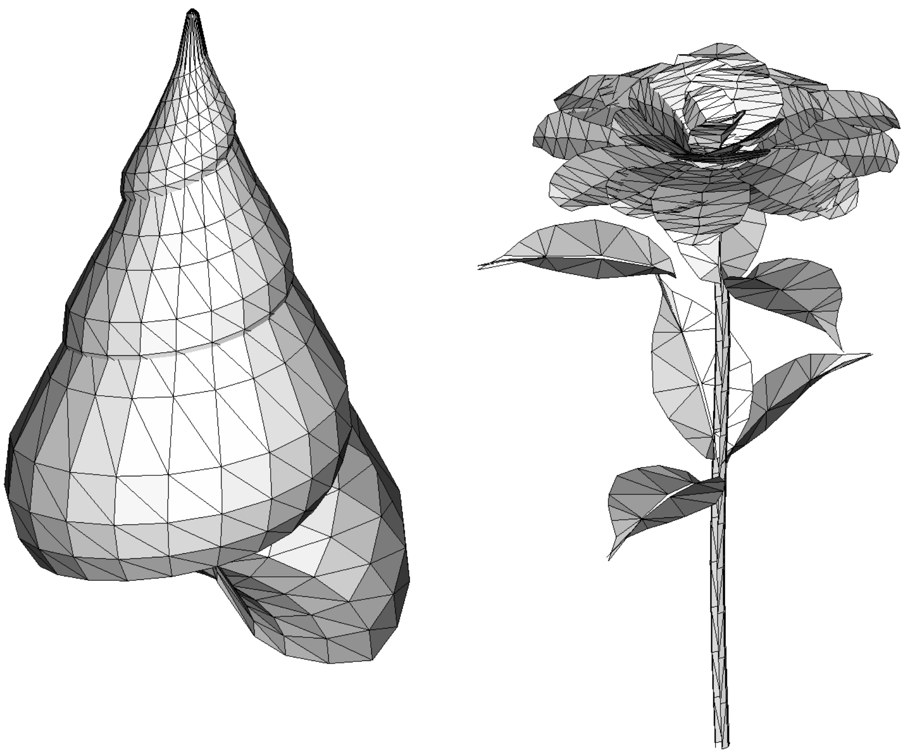


Figure 5-3: Examples of triangle soup

5.2 Closed polyhedra

On the other hand, closed polyhedra have several limitations and preprocessing is slow, but performance of collision detection is relatively fast. There are two types of closed polyhedra. One type is convex polyhedron (Shown in Figure 5-4).

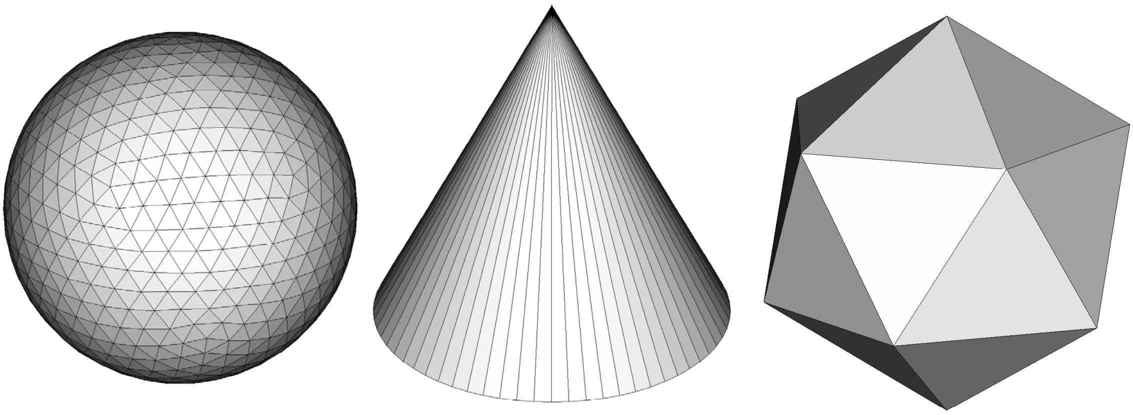


Figure 5-4: Examples of convex polyhedra

The other type is non-convex polyhedron (Shown in Figure 5-5). Convex polyhedra do not need preprocessing but non-convex polyhedron needs preprocessing. The types of preprocessing done on non-convex polyhedron is **convex surface decomposition** and **bounding volume hierarchy** of convex hulls.

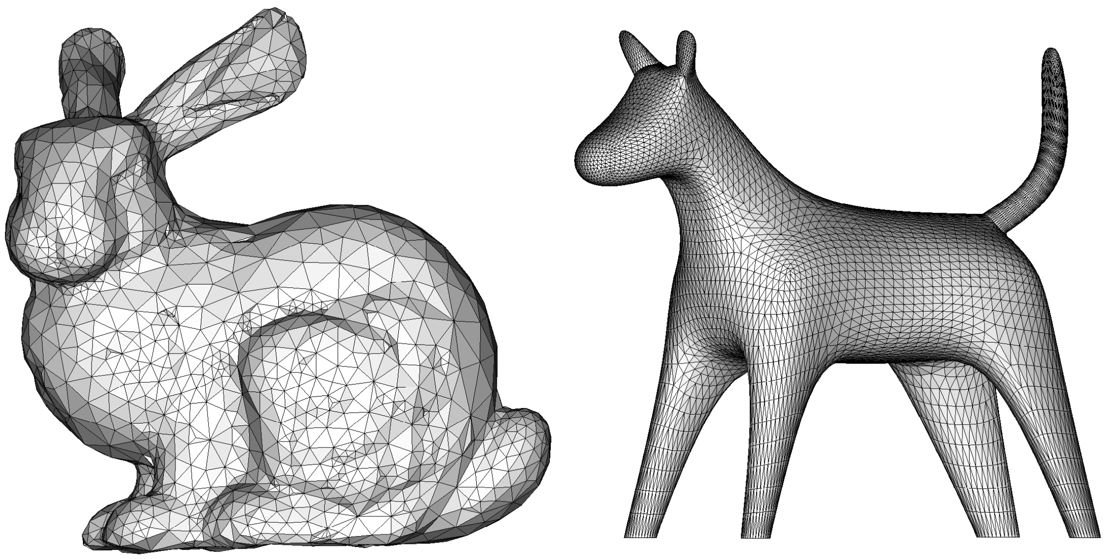


Figure 5-5: Examples of non-convex polyhedra

Closed polyhedra must of course be closed. Figure 5-6 shows an example of an unclosed polyhedron. In Figure 5-6, the edges, which are not shared by two triangles are red.

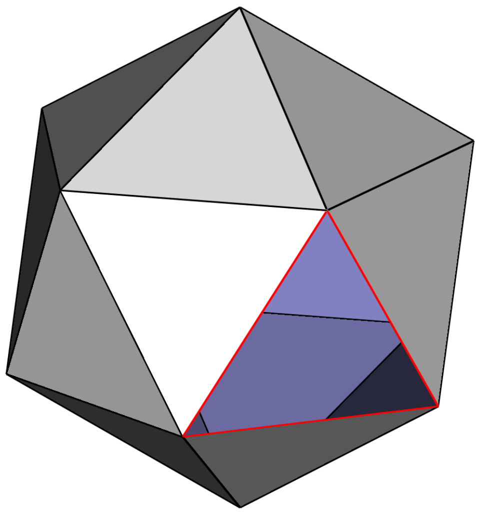


Figure 5-6: An example of an unclosed polyhedron

Closed polyhedra have inside and outside. Figure 5-7 shows the normal direction of a triangle. We define the negative direction of its normal to be the inside of the closed polyhedra, and the positive direction of normal is outside of closed polyhedra. If positive direction of the normals is a closed area according to the triangle winding, the normal direction of triangles of input geometry are inverted so that the positive normal point outside.

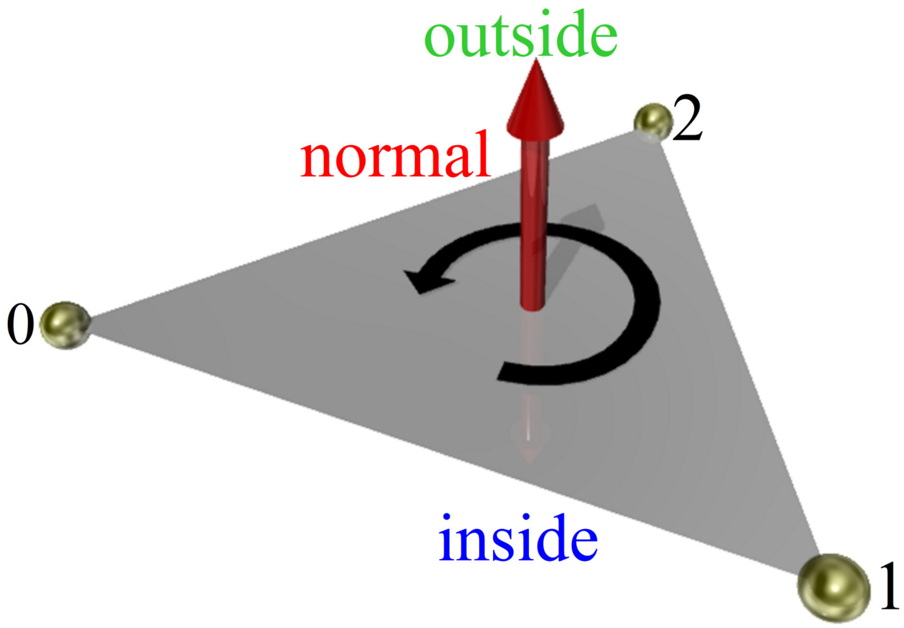


Figure 5-7: Normal direction

‘Closed’ means all edges are shared by two triangles and the directions of the edges are of an opposite winding. Figure 5-8 show an example of an edge shared by two triangles. In this example, the edge (101,102) is used in triangle A and B and the direction of the edge winding in triangle A is (101→102), and the direction of the edge in triangle B is (102→101).

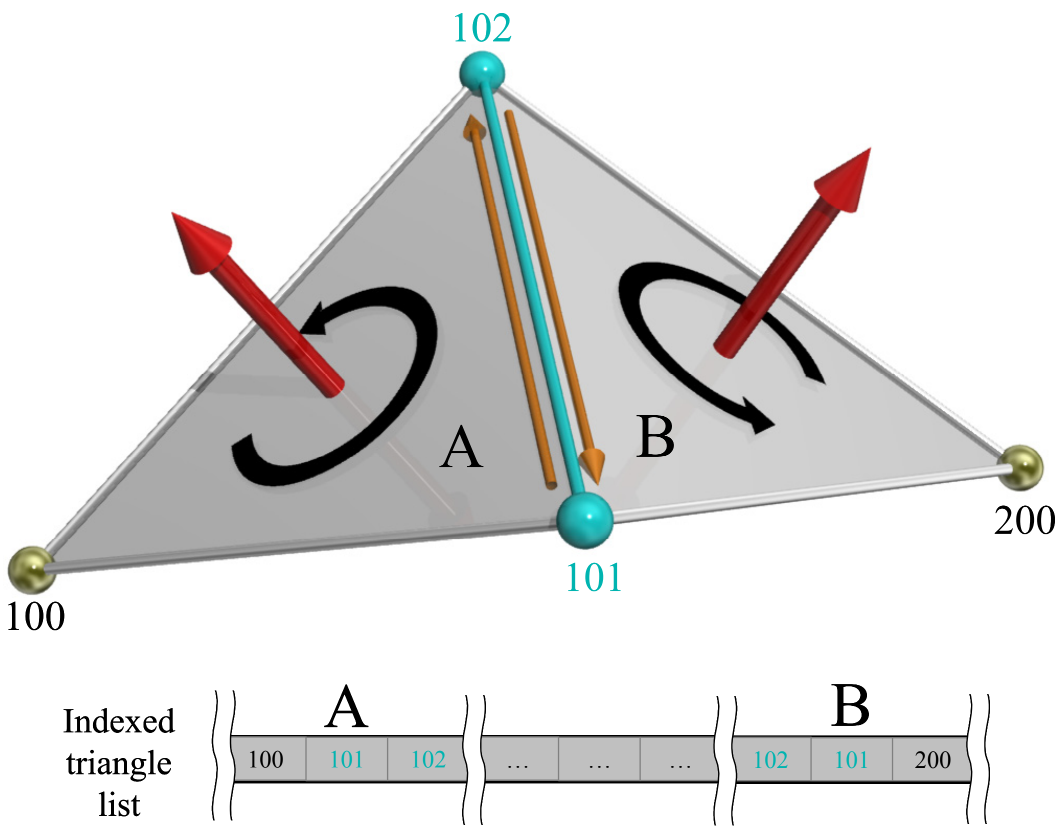


Figure 5-8: An example of an edge shared by two triangles

Figure 5-9 show normal vectors of a closed polyhedron.

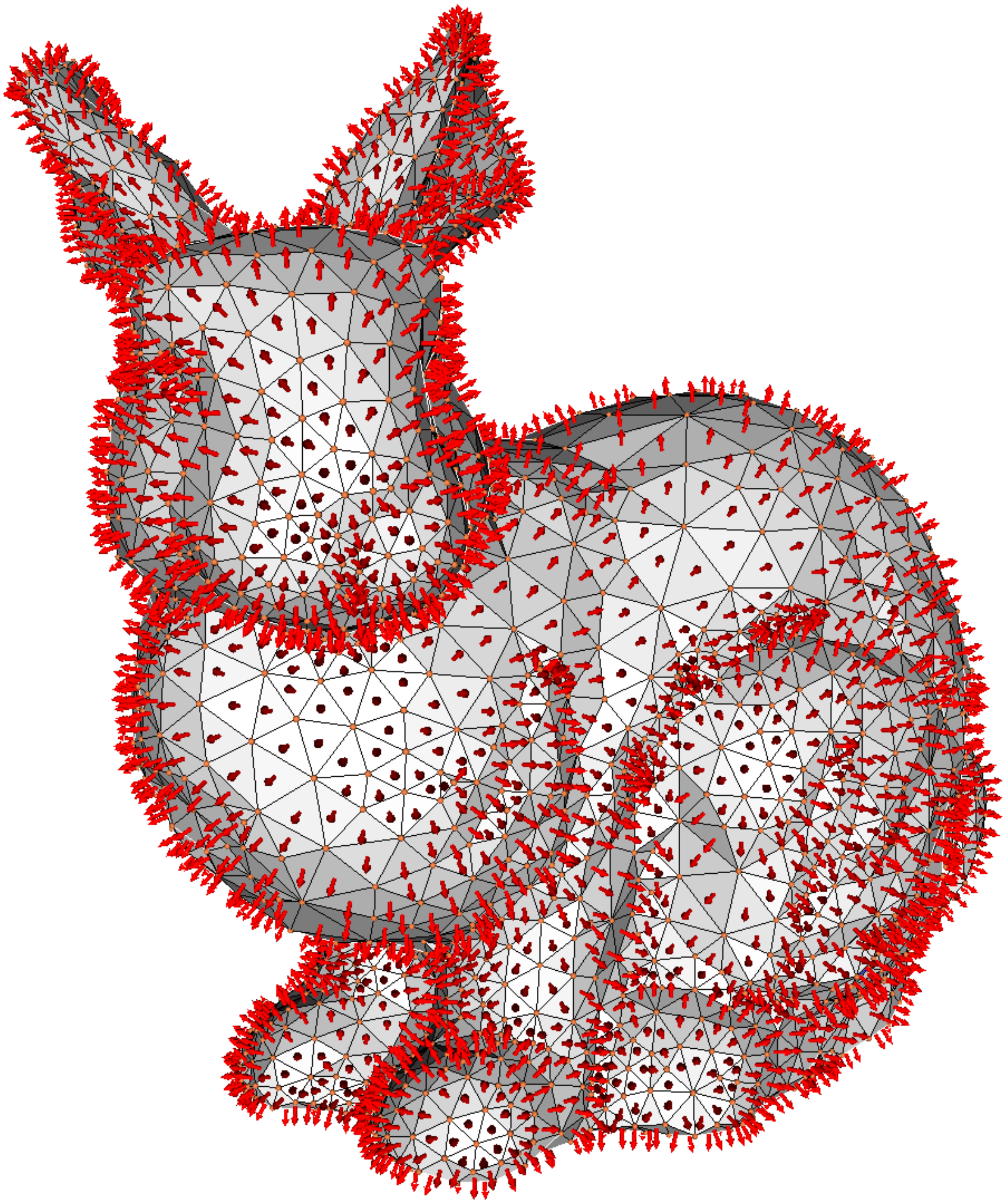
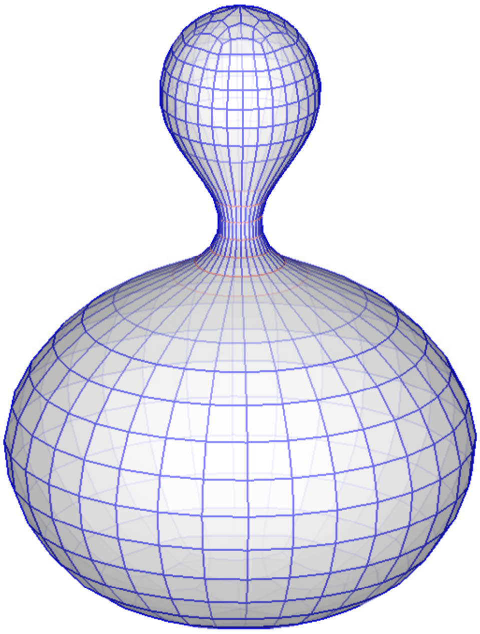
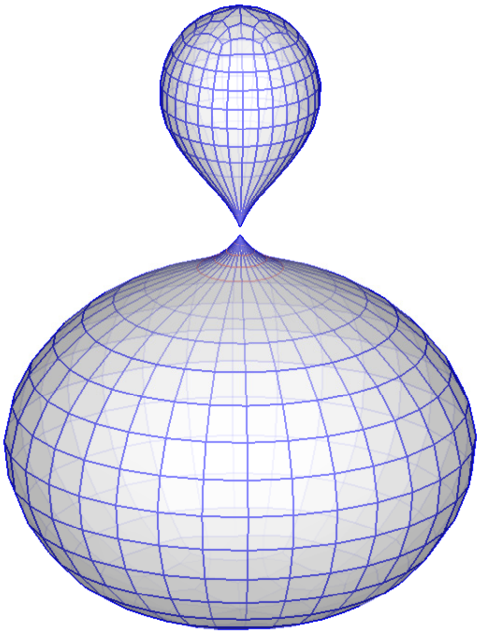


Figure 5-9: Normal vectors of a closed polyhedron

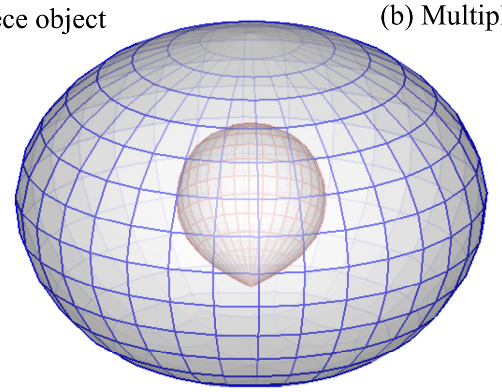
Triangles must be added to a SCObject as a **single-boundary piece** at a time. A single boundary piece is surrounded by only one boundary. On the other hand, a **multiple-boundary piece** is surrounded by multiple boundaries. Figure 5-10 (a) shows an example of single boundary piece and Figure 5-10 (b) shows an example of multiple-boundary piece. Figure 5-10 (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 5-10: Single/Multiple-piece object

One object can consist of multiple closed polyhedra. Figure 5-11 shows examples of compound convex polyhedra which are made from multiple convex polyhedra. Figure 5-12 shows examples of compound closed polyhedra. Closed polyhedra can have intersections in one object.

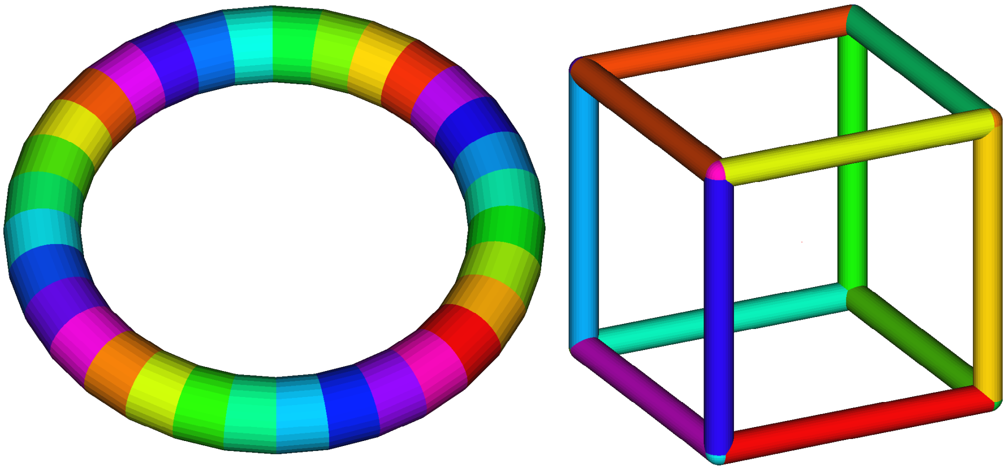


Figure 5-11: Examples of compound convex polyhedra

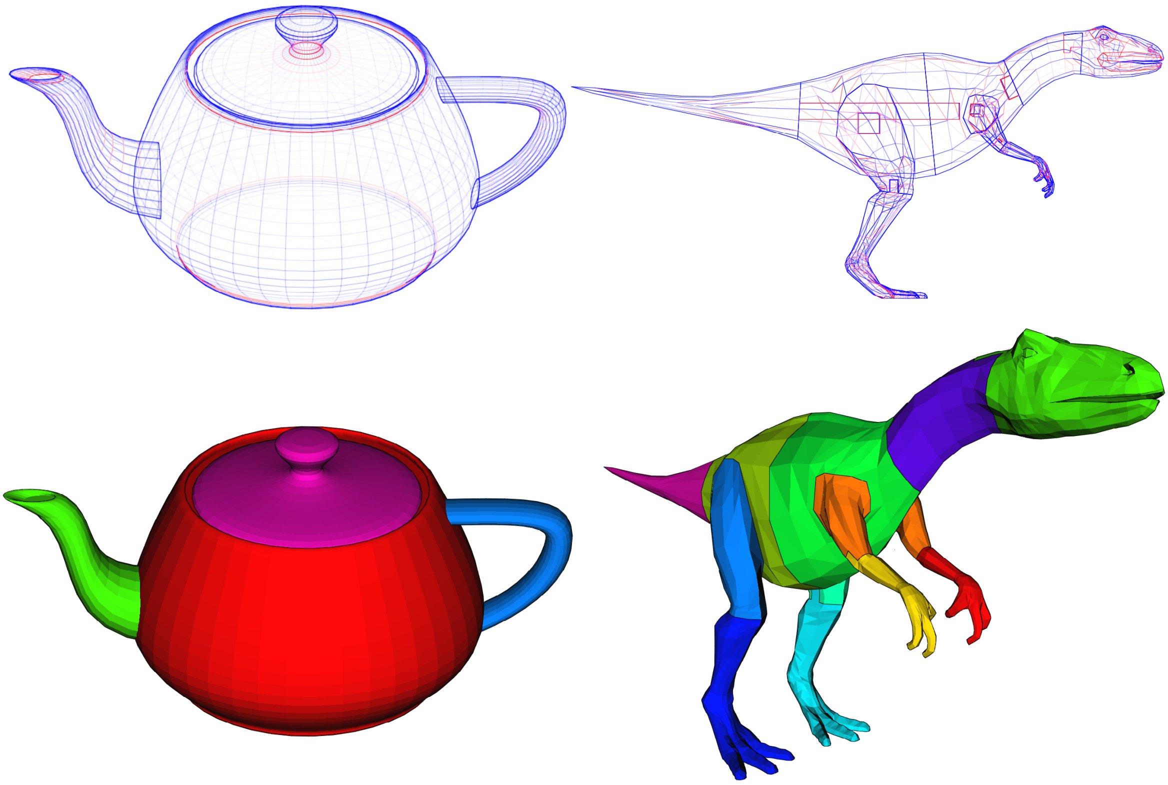


Figure 5-12: Examples of compound closed polyhedra

5.3 Convex surface decomposition

Non-convex polyhedra need preprocessing. SmartCollisionSDK uses convex surface decomposition to deal with collision detection between non-convex polyhedra. Figure 5-13 shows examples of convex surface decomposition. The result of convex surface decomposition consist of convex pieces. Convex piece may consist of only two triangle which have same triangle with opposite normal. Finally, BVH (Bounding Volume Hierarchies) using convex hull are built from the result of convex surface decomposition to accelerate collision detection as shown in Figure 5-14.

Figure 5-13: Examples of convex surface decomposition.

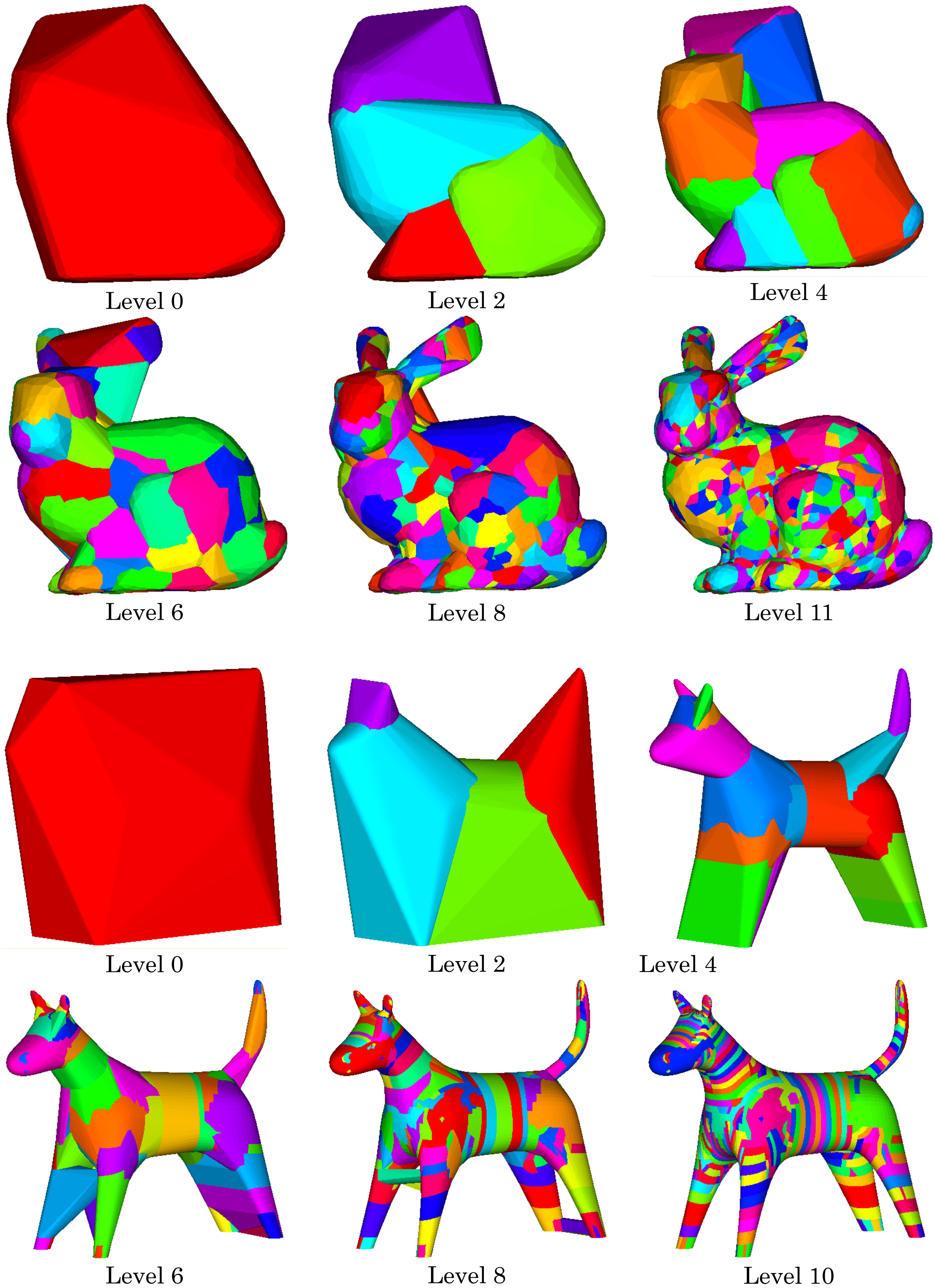


Figure 5-14: Examples of convex hull hierarchies

5.4 Format of input geometry

Both triangle soup and closed polyhedron use same format of input geometry. Figure 5-15 shows the format of an **indexed triangle set**. Indices of vertices start at 0.

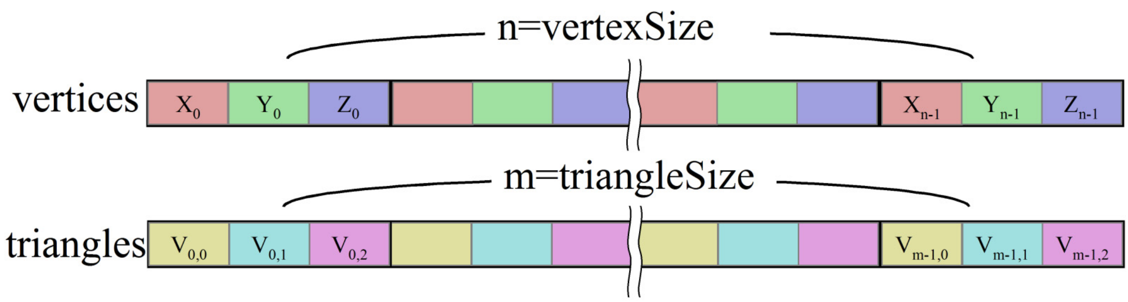


Figure 5-15: Indexed triangle set

5.5 How to give geometry data to SCObject

There are two types input geometry. Therefore, there are two types of SCObject for each types of input.

List 5-2 shows how to make SCObject for triangle soup.

List 5-1: How to make SCObject for triangle soup

```
SCObject object(SC_OBJECT_TYPE_TRIANGLE_SOUP);
```

List 5-2 shows how to make SCObject for closed polyhedra.

List 5-2: How to make SCObject for closed polyhedra

```
SCObject object(SC_OBJECT_TYPE_CLOSED_POLYHEDRA);
```

List 5-3 shows how to set geometry of a SCObject. Figure 5-16 shows the geometry of this example.

List 5-3: How to set goemetry

```
SCdouble 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
};
SCint triangles[3*4]={
    0,2,1, // triangle 0
    1,3,0, // triangle 1
    0,3,2, // triangle 2
    1,2,3  // triangle 3
};

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

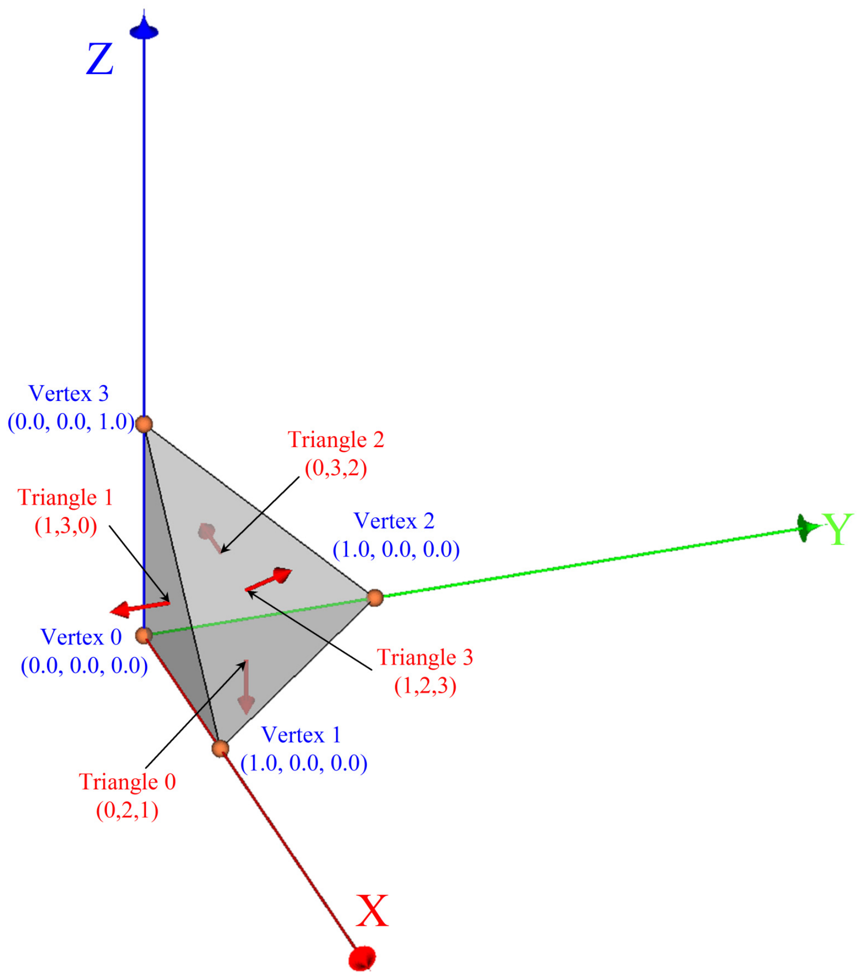


Figure 5-16: An example of geometry

It is possible to call SCObject::AddTriangles multiple times for an object consisting of multiple pieces.

List 5-4: How to make the object consisting of multiple pieces.

```
SCObject object(SC_OBJECT_TYPE_CLOSED_POLYHEDRON);

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
}
```

If the type of the object is SC_OBJECT_TYPE_CLOSED_POLYHEDRON and input geometry is non-convex polyhedron, pre-processing might take long time to make its BVH. It is possible to save BVHs and reuse them. List 5-5 shows how to make a BVH file and reuse it. In this example, if “test.bvh” does not exist, pre-processing is performed and the results of pre-processing is stored in “test.bvh”. If “test.bvh” exists, instead of performing pre-processing, the results of pre-processing is loaded from “test.bvh”.

List 5-5: How to make and reuse BVH

```
SCObject object(SC_OBJECT_TYPE_CLOSED_POLYHEDRON);

char bvhFile[]=”test.bvh”;

If(object.SetTriangles(vertex,vertexCount,triangles,triangleCount,bvhFile)!=SC_NO_ERROR){
    // Input geometry is invalid
}
```

6. Coordinate systems and transformation

There are two kind of coordinate system, namely world coordinate system and local coordinate system. World coordinate system is used to describe transformations of objects. Local coordinate system is used to describe geometry of object.

Figure 6-1 shows the world coordinate system and a local coordinate system. Center of rotation means the position in which the object rotates.

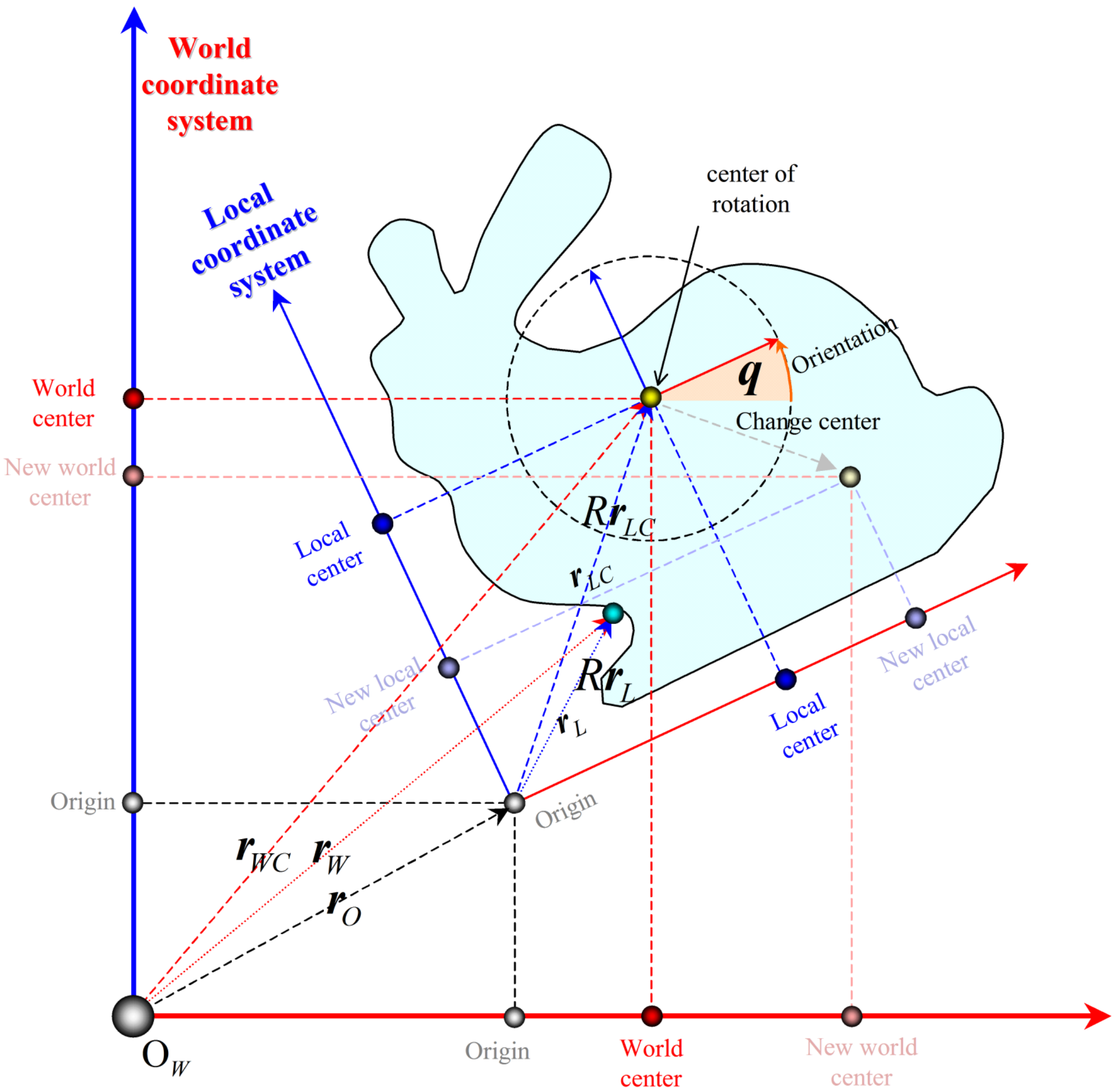


Figure 6-1: Coordinate systems

Let $\boldsymbol{r}_W$ a vector described in world coordinate system, and let $\boldsymbol{r}_L$ a vector described in local coordinate system, then relationship between $\boldsymbol{r}_W$ and $\boldsymbol{r}_L$ can be written by (6-1).

$$\boldsymbol{r}_W = R(\boldsymbol{r}_L - \boldsymbol{r}_{LC}) + \boldsymbol{r}_{WC} = R\boldsymbol{r}_L + \boldsymbol{r}_O = M\boldsymbol{r}_L \quad \cdots\cdots\cdots\cdots \quad (6\text{-}1)$$

Here,

$$M = T_{WC} R T_{LC}^{-1}$$

$$\boldsymbol{r}_W = \begin{pmatrix} x_W \\ y_W \\ z_W \\ 1 \end{pmatrix}, \boldsymbol{r}_L = \begin{pmatrix} x_L \\ y_L \\ z_L \\ 1 \end{pmatrix}$$

$$M = \begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

$$\boldsymbol{q} = \begin{pmatrix} s \\ v_x \\ v_y \\ v_z \end{pmatrix}, R = \begin{pmatrix} 1-2(v_y v_y + v_z v_z) & 2(v_x v_y - s v_z) & 2(v_x v_z + s v_y) & 0 \\ 2(v_x v_y + s v_z) & 1-2(v_z v_z + v_x v_x) & 2(v_y v_z - s v_x) & 0 \\ 2(v_x v_z - s v_y) & 2(v_y v_z + s v_x) & 1-2(v_x v_x + v_y v_y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\boldsymbol{r}_{WC} = \begin{pmatrix} x_{WC} \\ y_{WC} \\ z_{WC} \\ 1 \end{pmatrix}, \boldsymbol{r}_{LC} = \begin{pmatrix} x_{LC} \\ y_{LC} \\ z_{LC} \\ 1 \end{pmatrix}, T_{WC} = \begin{pmatrix} 1 & 0 & 0 & x_{WC} \\ 0 & 1 & 0 & y_{WC} \\ 0 & 0 & 1 & z_{WC} \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_{LC} = \begin{pmatrix} 1 & 0 & 0 & x_{LC} \\ 0 & 1 & 0 & y_{LC} \\ 0 & 0 & 1 & z_{LC} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

M means transformation matrix. $\boldsymbol{q}$ means orientation described by quaternion. R means orientation described by matrix. $\boldsymbol{r}_{WC} / \boldsymbol{r}_{LC}$ means world/local center of rotation described by a vector. $\boldsymbol{r}_O$ means the origin of local coordinate system. T_{WC} / T_{WC} means world/local center of rotation described by a matrix.

Vector and quaternion and matrix are stored in arrays as shown in Figure 6-2.

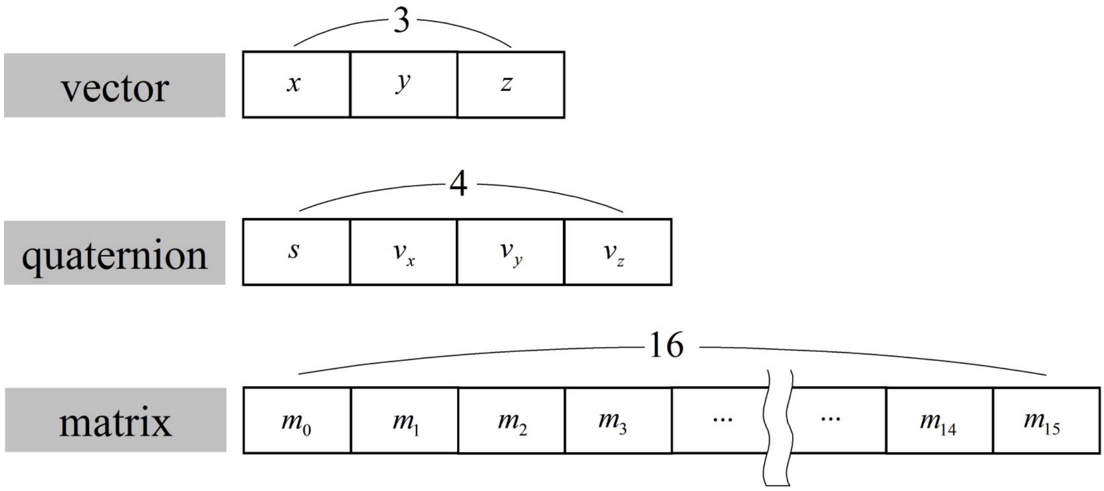


Figure 6-2: Arrays of vector and quaternion and matrix

SCObject internally stores $\boldsymbol{r}_{WC}, \boldsymbol{r}_{LC}$ and $\boldsymbol{q}$ to specify its transformation.

If a SCObject has been not added to SCSceneManager, the transformations of the SCObject can be set by using the method SCObject::SetTransformation. After a SCObject has been added to SCSceneManager, it is required that you use the method SCSceneManger::SetTransformation to set the transformations of the SCObject (See 7.2). The syntax of SCObject::SetTransformation are as follows.

```
SCint SCObject::SetTransformation(SCenum type, const SCdouble*trans);
SCint SCObject::SetTransformation(SCenum type, const SCfloat*trans);
```

SetTransformation has an argument to specify types of transformation. Table 6-1 show types of transformation. The size of array for the values depends on the type of transformation.

Table 6-1: Types of SetTransformation

Types of transformation	Input value	Value to be changed	Value to be set
SC_POSITION_ORIGIN	$\boldsymbol{r}'_O$	$\boldsymbol{r}_{WC}$	$\boldsymbol{r}'_O + R\boldsymbol{r}_{LC}$
SC_POSITION_WORLD_CENTER	$\boldsymbol{r}'_{WC}$	$\boldsymbol{r}_{WC}$	$\boldsymbol{r}'_{WC}$
SC_POSITION_LOCAL_CENTER	$\boldsymbol{r}'_{LC}$	$\boldsymbol{r}_{WC}$	$R(\boldsymbol{r}'_{LC} - \boldsymbol{r}_{LC}) + \boldsymbol{r}_{WC}$
SC_POSITION_NEW_WORLD_CENTER	$\boldsymbol{r}'_{WC}$	$\boldsymbol{r}_{LC}$	$R^{-1}(\boldsymbol{r}'_{WC} - \boldsymbol{r}_{WC}) + \boldsymbol{r}_{LC}$
		$\boldsymbol{r}_{WC}$	$\boldsymbol{r}'_{WC}$
SC_POSITION_NEW_LOCAL_CENTER	$\boldsymbol{r}'_{LC}$	$\boldsymbol{r}_{LC}$	$\boldsymbol{r}'_{LC}$
		$\boldsymbol{r}_{WC}$	$R(\boldsymbol{r}'_{LC} - \boldsymbol{r}_{LC}) + \boldsymbol{r}_{WC}$
SC_ORIENTATION_QUATERNION	$\boldsymbol{q}'$	$\boldsymbol{q}$	$\boldsymbol{q}'$
SC_ORIENTATION_MATRIX	R'	$\boldsymbol{q}$	$\boldsymbol{q}'(R')$
SC_TRANSFORMATION_MATRIX	M'	$\boldsymbol{r}_{WC}$	$\boldsymbol{r}'_{WC}(M')$
		$\boldsymbol{q}$	$\boldsymbol{q}'(M')$

SC_POSITION_NEW_WORLD_CENTER and SC_POSITION_NEW_LOCAL_CENTER change the local center of rotation ($\boldsymbol{r}_{LC}$) without changing the origin of local coordinate system($\boldsymbol{r}_O$).

Transformation of SCObject can be obtained by using the method SCObject::GetTransformation. The syntax of SCObject::GetTransformation is as follows.

```
SCint SCObject::GetTransformation(SCenum type, SCdouble*trans)const;
SCint SCObject::GetTransformation(SCenum type, SCfloat*trans)const;
```

GetTransformation also has an argument to specify types of transformation. Table 6-2 shows types of GetTransforamtion. The size of array for the values depends on the type of transformation.

Table 6-2: Types of GetTransformation

Types of transformation	Value to be get
SC_POSITION_ORIGIN	$\boldsymbol{r}_{WC} - R\boldsymbol{r}_{LC}$
SC_POSITION_WORLD_CENTER	$\boldsymbol{r}_{WC}$
SC_POSITION_LOCAL_CENTER	$\boldsymbol{r}_{LC}$
SC_ORIENTATION_QUATERNION	$\boldsymbol{q}$
SC_ORIENTATION_MATRIX	R
SC_TRANSFORMATION_MATRIX	M

List 6-1 and List 6-2 show how to set transformation of SCObject. Figure 6-3 and Figure 6-4 show transition of transformation of the object. The resulting transformations are the same.

List 6-1: How to set transformation(1)

```
SCdouble local_center[3]={13,6,11};
SCdouble world_center[3]={10,50,35};
SCdouble orientation[4]={0.707107,0.707107,0,0};// 90 degree rotation around x axis

SCObject object(SC_OBJECT_TYPE_CLOSED_POLYHEDRON);                    // (1)
object.SetTransformation(SC_POSITION_NEW_LOCAL_CENTER, local_center); // (2)
object.SetTransformation(SC_POSITION_WORLD_CENTER,world_center);      // (3)
object.SetTransformation(SC_ORIENTATION_QUATERNION,orientation);      // (4)
```

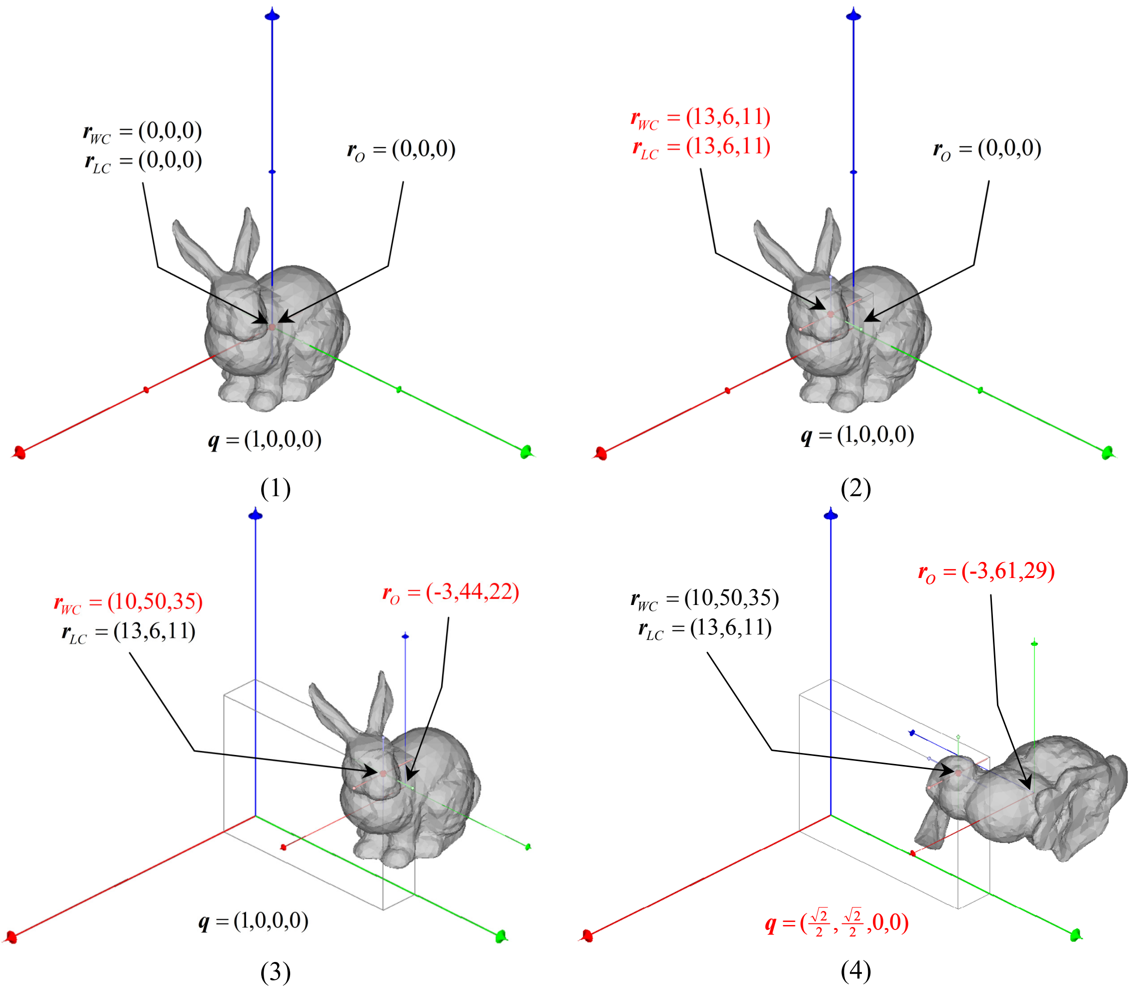


Figure 6-3: Transition of transformation(1)

List 6-2: How to set transformation(2)

```
SCdouble origin[3]={-3,61,29};
SCdouble world_center[3]={10,50,35};
SCdouble orientation[4]={0.707107,0.707107,0,0};// 90 degree rotation around x axis

SCObject object(SC_OBJECT_TYPE_CLOSED_POLYHEDRON);                          // (1)
object.SetTransformation(SC_POSITION_ORIGIN,origin);                        // (2)
object.SetTransformation(SC_ORIENTATION_QUATERNION,orientation);            // (3)
object.SetTransformation(SC_POSITION_NEW_WORLD_CENTER,world_center);        // (4)
```

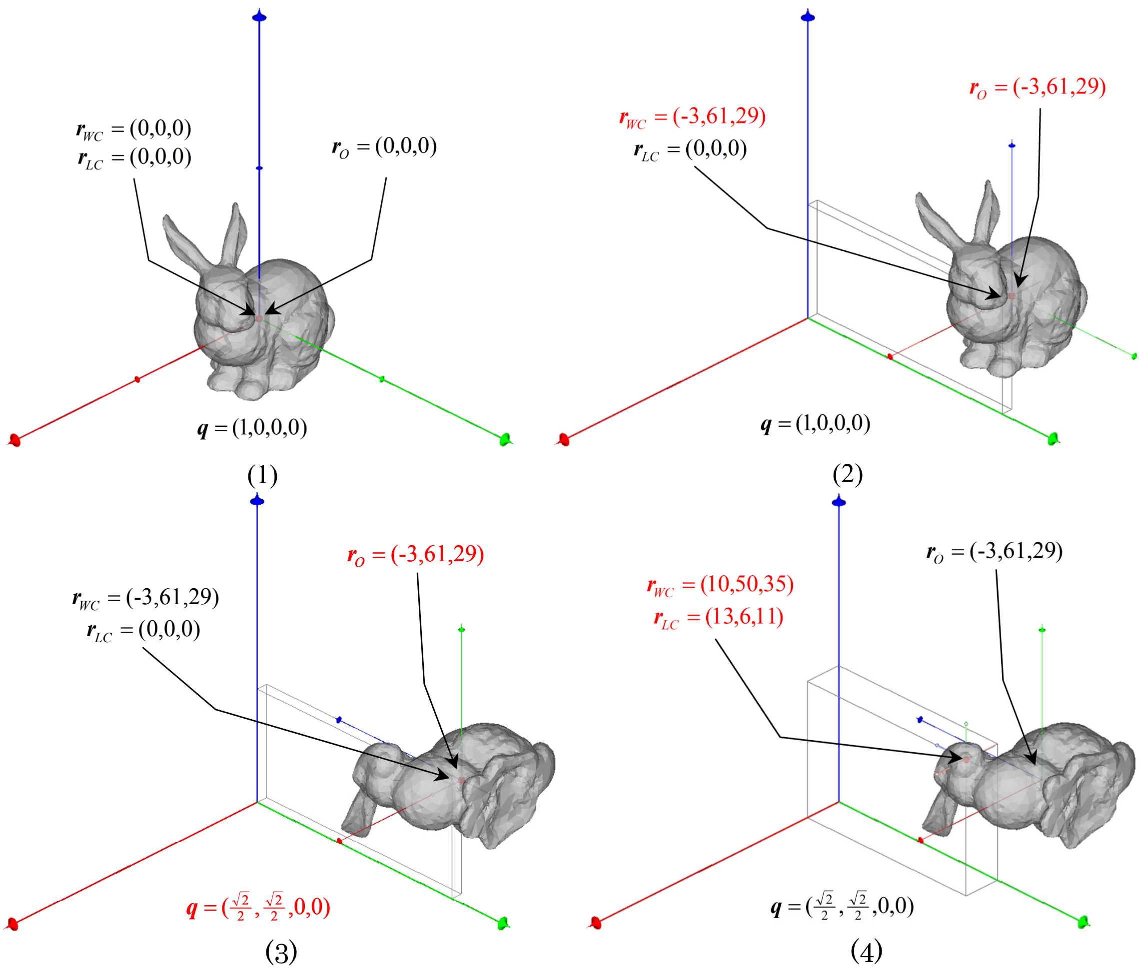


Figure 6-4: Transition of transformation(2)

7. Collision detection

Collision detections are performed by SCSceneManager. There are two types of SCSceneManager for each type of input geometry.

List 7-1 shows how to construct SCSceneManager for triangle soup.

List 7-1: How to construct SCSceneManager for triangle soup

```
SCSceneManager scene(SC_SCENE_MANAGER_TRIANGLE_SOUP);
```

List 7-2 shows how to construct SCSceneManager for closed polyhedra.

List 7-2: How to construct SCSceneManager for closed polyhedra

```
SCSceneManager scene(SC_SCENE_MANAGER_CLOSED_POLYHEDRA);
```

SCSceneManager accepts only same types of input geometry, and it is not allowed that mixture of difference types of input geometry in a scene.

7.1 Setup of objects and groups

Objects in the scene have unique IDs which are assigned by the application developer when the objects are added to the scene. Syntax of the methods to add objects to the scene is as follows.

```
SCint SCObject::AddObject(SCint id, SCObject*object);
```

Here, *id* is the ID which are assigned to *object.* List 7-3 shows how to add objects to the scene.

List 7-3: How to add objects to the scene.

```
SCSceneManager scene(SC_SCENE_MANAGER_CLOSED_POLYHEDRA);
SCObject object1(SC_OBJECT_TYPE_CLOSED_POLYHEDRA);
SCObject object2(SC_OBJECT_TYPE_CLOSED_POLYHEDRA);
SCObject object3(SC_OBJECT_TYPE_CLOSED_POLYHEDRA);
SCObject object4(SC_OBJECT_TYPE_CLOSED_POLYHEDRA);
SCObject object5(SC_OBJECT_TYPE_CLOSED_POLYHEDRA);
SCObject object6(SC_OBJECT_TYPE_CLOSED_POLYHEDRA);
// Add triangles for each SCObject
…
scene.AddObject(0,&object1);
scene.AddObject(1,&object2);
scene.AddObject(2,&object3);
scene.AddObject(3,&object4);
scene.AddObject(4,&object5);
scene.AddObject(5,&object6);
```

Objects in the scene can be deleted. Syntax of the methods to delete objects from the scene is as follows.

```
SCint SCObject::DeleteObject(SCint id);
```

List 7-4 shows how to delete objects from the scene.

List 7-4: How to delete objects from the scene.

```
…
scene.DeleteObject(4);
scene.DeleteObject(5);
```

Each object belongs to a group. Groups in the scene also have unique IDs, which are distinct from the IDs given to objects. By default, the objects are automatically added to the **global static group**. The ID of the global static group is a negative integer and defined by SC_GROUP_STATIC. IDs which are given by users must be positive integers. It is not possible to add objects to a global static group explicitly. The IDs of groups are also given by using the following method.

```
SCint SCObject::AddObjectToGroup(SCint gid, SCint id);
```

Here, *gid* is the ID of the group to which the object specified by *id* is added. List 7-5 shows how to add objects to groups.

List 7-5: How to add objects to groups.

```
…
scene.AddObjectToGroup(0,0);
scene.AddObjectToGroup(0,1);
scene.AddObjectToGroup(1,2);
scene.AddObjectToGroup(1,3);
```

The objects in the groups can be deleted using the following method.

```
SCint SCObject::DeleteObjectFromGroup(SCint gid, SCint id);
```

List 7-6 Shows how to delete objects from groups.

List 7-6: How to delete objects from groups.

```
…
scene.DeleteFromGroup(0,1);
scene.DeleteFromGroup(1,3);
```

Objects deleted from their groups are automatically returned to a global static group.

7.2 Setup of transformations

After a SCObject is added in SCSceneManager, SCObject::SetTransforamtion can not be used to set its transformation. Therefore it is required that you use the method SCSceneManger::SetTransformation to set the transformations of the SCObject after adding it to the SCSceneManager. SCSceneManger::SetTransformation works not for each object but for each group in the scene. The local coordinate system of a group is inherited from the first object added to the group, and the transformations of other objects relative to the local coordinate system are constant after being added to the group.

The syntax of SCSceneManger::SetTransformation is as follows.

```
SCint SCSceneManager::SetTransformation(SCint gid,SCenum type, const SCdouble*trans);
SCint SCSceneManager::SetTransformation(SCint gid,SCenum type, const SCfloat*trans);
```

gid specifies the ID of the group, and *type* specifies the type of transformation. Table 6-1 show types of transformation. List 7-7 shows how to set transformations of groups.

List 7-7: How to set transformations for groups.

```
SCdouble position[3]={100.0, 200.0, -150.0};
SCdouble center1[3]={50.0, 100.0, -75.0};
SCdouble center2[3]={100.0, 100.0, 100.0};
SCdouble orientation[4]={1.0, 0.0, 0.0, 0.0};
SCdouble matrix[16]={1.0, 0.0, 0.0, 0.0,
                     0.0, 1.0, 0.0, 0.0,
                     0.0, 0.0, 1.0, 0.0,
                     50.0, -30.0, 100.0, 1.0};

scene.SetTransformation(0,SC_NEW_WORLD_CENTER,center1);
scene.SetTransformation(0,SC_POSITION_WORLD_CENTER,position);
scene.SetTransformation(0,SC_ORIENTATION_QUATERNION,orientation);

scene.SetTransforamtion(1,SC_TRANSFORMATION_MATRIX,matrix);
scene.SetTransformation(1,SC_NEW_WORLD_CENTER,center2);
```

Transformation of each group can be obtained by using the SCManager::GetTransformation method. The syntax of SCManager::GetTransformation is as follows.

```
SCint SCSceneManager::GetTransformation(SCint gid,SCenum type, SCdouble*trans)const;
SCint SCSceneManager::GetTransformation(SCint gid,SCenum type, SCfloat*trans)const;
```

Table 6-2 shows the types of transforamtion. List 7-8 shows how to set transformations for groups.

List 7-8: How to get transformations of groups.

```
SCdouble position[3];
SCdouble orientation[4];
SCdouble matrix[16];

scene.GetTransformation(0,SC_POSITION_WORLD_CENTER,position);
scene.GetTransformation(0,SC_ORIENTATION_QUATERNION,orientation);
scene.GetTransforamtion(1,SC_TRANSFORMATION_MATRIX,matrix);
```

It is possible to use SCObject::GetTransformation to get transformations for each object.

7.3 Target and its opponent

Collision detections are performed between pairs of groups. One of the groups is called '**target**' and the other is called '(its) **opponent**'.

7.4 Activity of group pairs

If there are N groups in the scene, there are $\frac{1}{2}N(N+1)$ possible pairs. In Figure 7-1, there are 7 objects, 3 groups, and 3 pairs of groups in the scene. By default, collision detections are performed with respect to all possible pairs of groups.

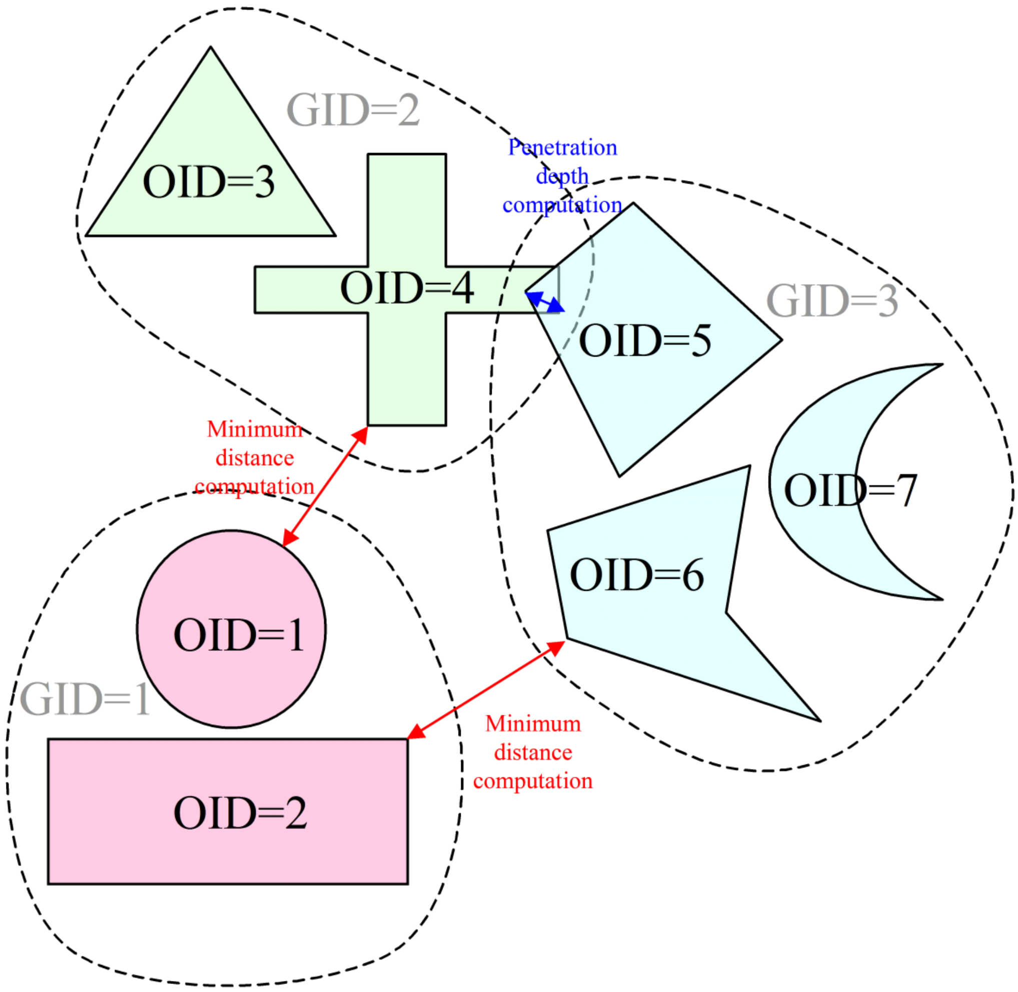


Figure 7-1: Activities of collision detection

The specific computations performed during the collision detection sequence can be controlled in terms of the **activity states** of objects, groups, and the pairs of groups.

There are 4 types of possible active states for a groups to be in: SC_ACTIVITY_ACTIVE, SC_ACTIVITY_SLEEPING, SC_ACTIVITY_PASSIVE, and SC_ACTIVITY_INACTIVE. Table 7-1 shows the direction of collision detection according to the activities of two groups. In Table 7-1, the group at the origin of the arrow plays the role of the **target**, and the group at the head of the arrow plays the role of the **opponent**.

- If both activities of two groups are SC_ACTIVITY_ACTIVE, both distance computation and penetration depth computation are performed bi-directionally between them.
- If one of the group pair is set to SC_ACTIVITY_ACTIVE and the other is set to SC_ACTIVITY_SLEEPING or SC_ACTIVITY_PASSIVE, distance computation and penetration depth computation are performed with respect to former object.
- If one of the activity states is SC_ACTIVITY_INACTIVE, collision detection is not performed between them.

Difference between SC_ACTIVITY_SLEEPING and SC_ACTIVITY_PASSIVE is that groups in former type of activity keep their collision status information with respect to any other group, and groups in later type of activity do not keep any collision status information. This affects penetration depth computation when activities of objects change from SC_ACTIVITY_SLEEPING to SC_ACTIVITY_ACTIVE, because penetration depth computation requires knowledge about the previous collision status of the group pair. The activity of the **global static group** is fixed to SC_ACTIVITY_PASSIVE.

Table 7-1: Activities of group pairs according to the their activity state

Activity of group A \ Activity of group B	SC_ACTIVITY_ACTIVE	SC_ACTIVITY_SLEEPING	SC_ACTIVITY_PASSIVE	SC_ACTIVITY_INACTIVE
SC_ACTIVITY_ACTIVE	A⇄B	A→B	A→B	—
SC_ACTIVITY_SLEEPING	A←B	—	—	—
SC_ACTIVITY_PASSIVE	A←B	—	—	—
SC_ACTIVITY_INACTIVE	—	—	—	—

Pairs of groups have 4 types of activities. SC_ACTIVITY_ACTIVE/SC_ACTIVITY_INACTIVE activates/deactivates bi-directionally. SC_ACTIVITY_ONE_WAY_ACTIVE/SC_ACTIVITY_ONE_WAY_INACTIVE activates/deactivates only one way.

Each object has 2 types of activities (SC_ACTIVITY_ACTIVE, SC_ACTIVITY_INACTIVE). If activity of the object is SC_ACTIVITY_INACTIVE, the object is not taken into account of collision detection. List 7-9 shows how to set activities of objects.

List 7-9: How to set activities of objects

```
…
scene.SetActivityObject(0,SC_ACTIVITY_ACTIVE);
scene.SetActivityObject(1,SC_ACTIVITY_INACTIVE);
```

List 7-10 shows how to set activities of group. Figure 7-2 shows resulting activities of group pairs according to List 7-10.

List 7-10: How to set activities of groups

```
…
scene.SetActivityGroup(0,SC_ACTIVITY_ACTIVE);
scene.SetActivityGroup(1,SC_ACTIVITY_PASSIVE);
scene.SetActivityGroup(2,SC_ACTIVITY_INACTIVE);
scene.SetActivityGroup(3,SC_ACTIVITY_SLEEPING);
scene.SetActivityGroup(4,SC_ACTIVITY_ACTIVE);
scene.SetActivityGroup(5,SC_ACTIVITY_SLEEPING);
scene.SetActivityGroup(6,SC_ACTIVITY_PASSIVE);
```

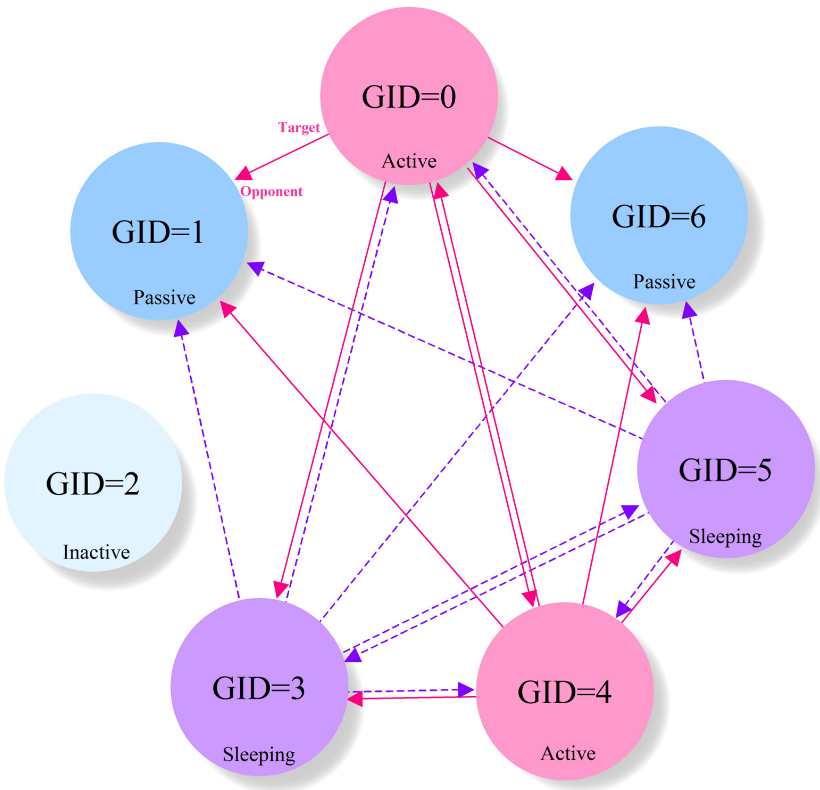
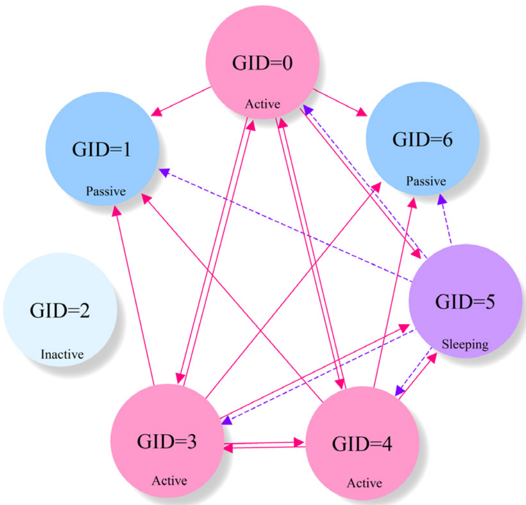


Figure 7-2: Resulting activities of group pairs according to group activity states

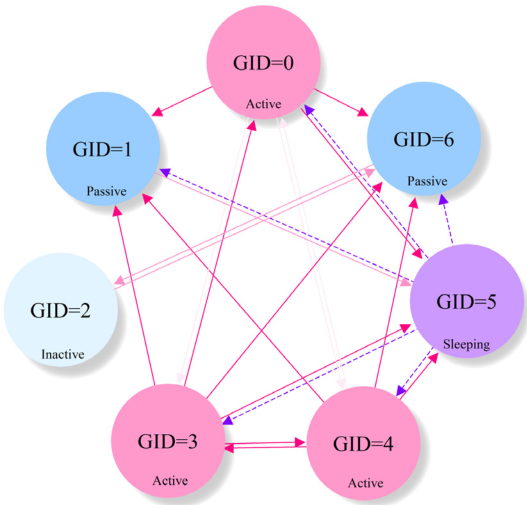
List 7-11 shows how to set activities of group pairs. The activities of group pairs specified directly are overwritten on activities according to group activity states. Figure 7-3 shows modified activities of group pairs according to List 7-11.

List 7-11: How to set activities of group pairs

```
…
scene.SetActivityGroup(0,SC_ACTIVITY_ACTIVE);
scene.SetActivityGroup(1,SC_ACTIVITY_PASSIVE);
scene.SetActivityGroup(2,SC_ACTIVITY_INACTIVE);
scene.SetActivityGroup(3,SC_ACTIVITY_ACTIVE);
scene.SetActivityGroup(4,SC_ACTIVITY_ACTIVE);
scene.SetActivityGroup(5,SC_ACTIVITY_SLEEPING);
scene.SetActivityGroup(6,SC_ACTIVITY_PASSIVE);
// Before set activities of group pairs
scene.SetActivityGroupPair(2,6,SC_ACTIVITY_ACTIVE);
scene.SetActivityGroupPair(0,4,SC_ACTIVITY_INACTIVE);
scene.SetActivityGroupPair(1,5,SC_ACTIVITY_ONE_WAY_ACTIVE);
scene.SetActivityGroupPair(0,3,SC_ACTIVITY_ONE_WAY_INACTIVE);
// After set activities of group pairs
```



(1)Before set activities of group pairs



(2)After set activities of group pairs

Figure 7-3: Modified activities of group pairs

7.5 Minimum distance computation

When there is no intersection between a pair of groups, the minimum distance computation is performed.

The results of minimum distance computation are a pair of points, and the minimum distance vector between them. A pair of points consists of the end point on the surface of target and the point on the surface of opponent. When there are multiple pairs of points which have the same distance, only one of them is obtained. The result of minimum distance computation is shown in Figure 7-4.

If the activities of both groups are SC_ACTIVITY_ACTIVE, the results of the minimum distance computation in which the roles of target and opponent are exchanged can also be obtained.

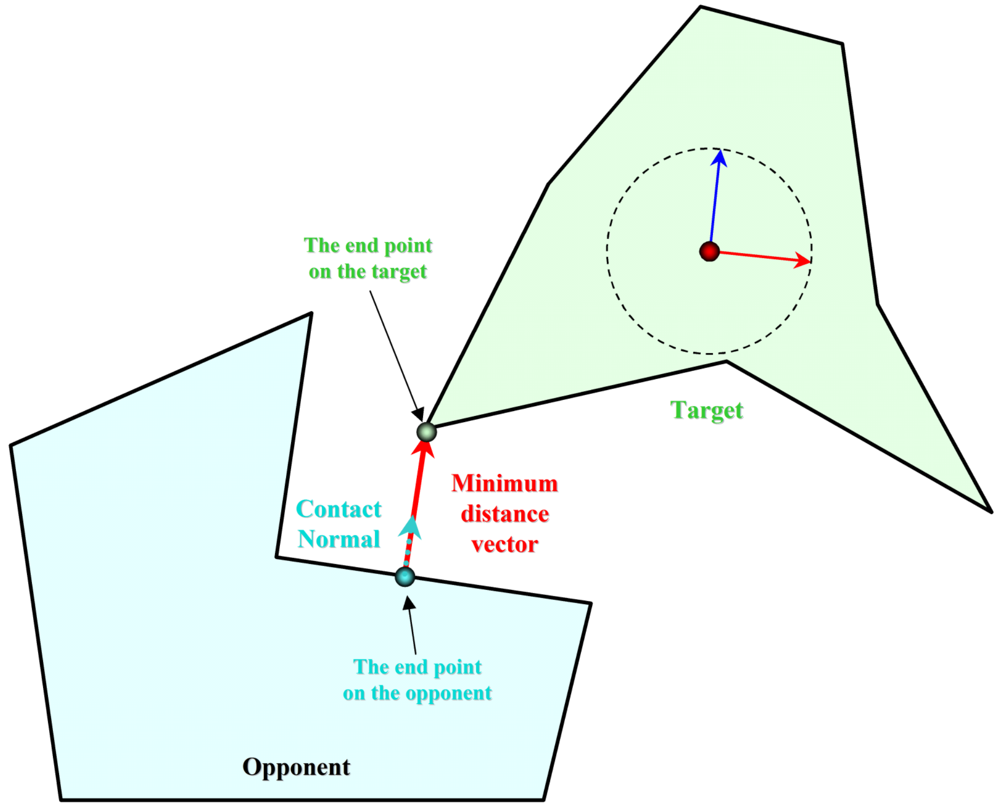


Figure 7-4: Minimum distance computation

Minimum distance computation is performed only if the distance between pairs of groups is smaller than a threshold value. This threshold value is the maximum distance of minimum distance computation. The application requires The maximum distance can be set as shown in List 7-12.

List 7-12: How to set the threshold of minimum distance computation

```
SCdouble maxDistance=10;// the threshold value of minimum distance computation
SCSceneManager scene(SC_SCENE_MANAGER_CLOSED_POLYHEDRA);
// Setting of transformation and attributes of SCSceneManager
…
scene.SetAttributeDouble(SC_SCENE_MANAGER_MAX_DISTANCE,maxDistance);
```

Minimum distance computation differs between geometry data type. Figure 7-5 and Figure 7-6 shows minimum distance computation of closed polyhedra (solid model) and triangle soup (surface model) respectively.

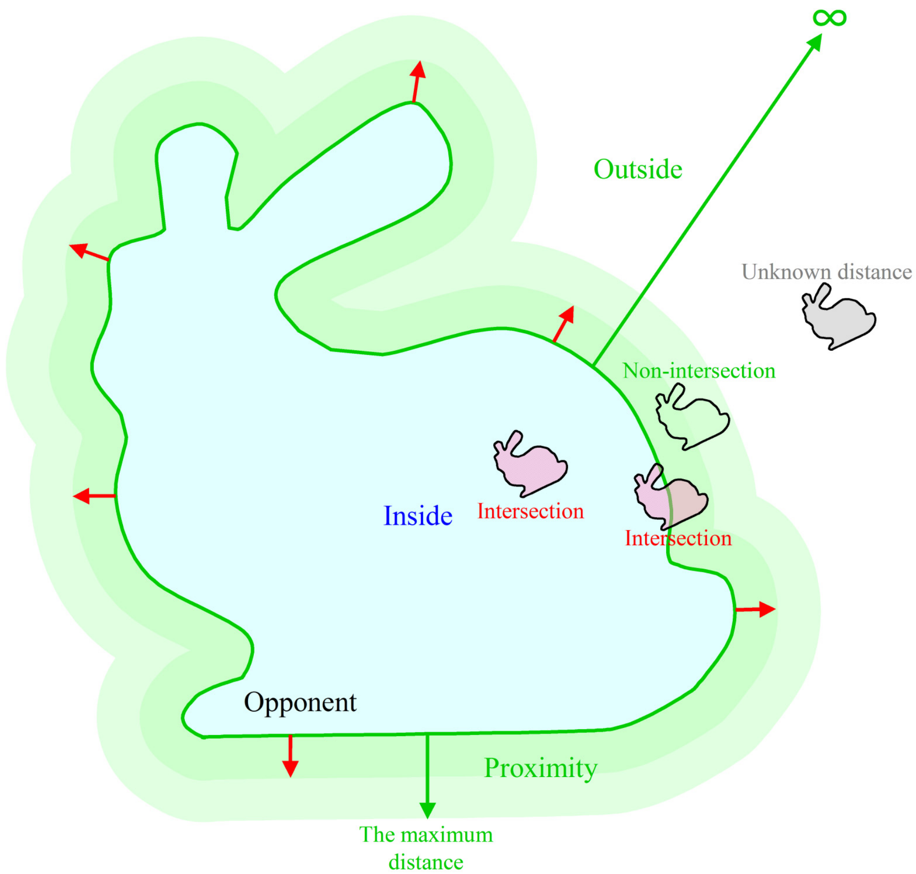


Figure 7-5: Minimum distance of closed Polyhedra (Solid model)

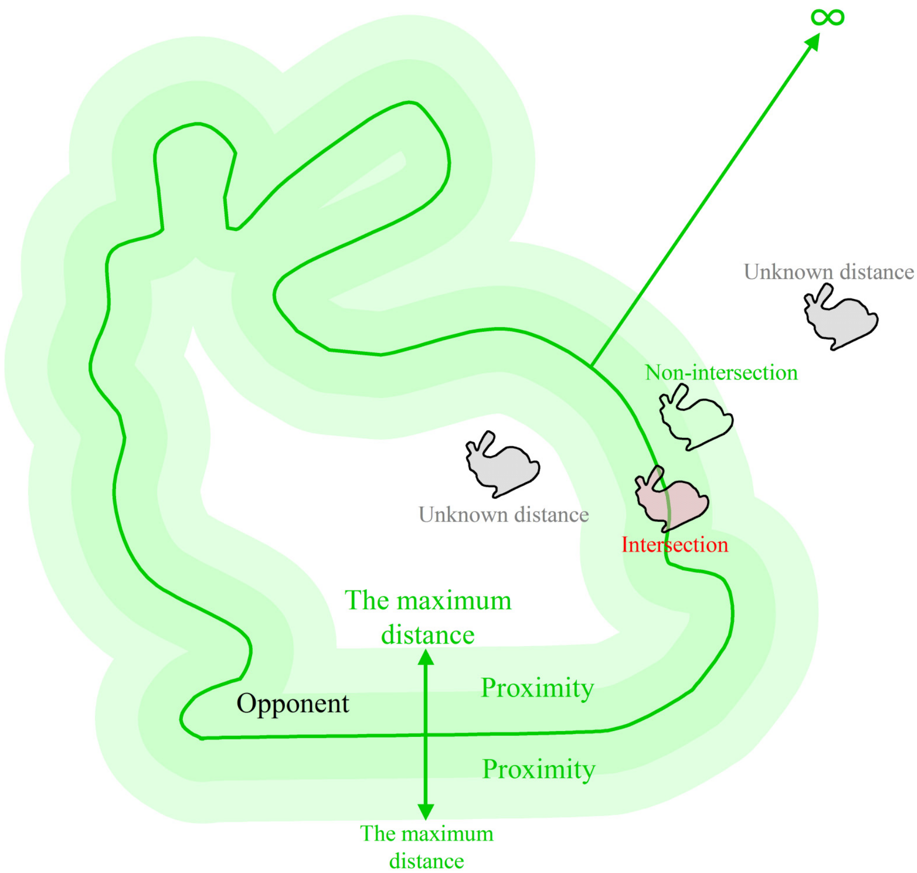


Figure 7-6: Minimum distance of triangle soup (Surface model)

7.6 Penetration depth computation

When the intersection between a pair of groups happens for the first time, penetration depth computation starts instead of the minimum distance computation, and continues as long as penetration depth is not zero. The penetration depth computation algorithm needs an initial solution, which is obtained from the result of minimum distance computation. Therefore, minimum distance computation must be performed before penetration depth computation.

The results of penetration depth computation are shown in Figure 7-7. The distance is zero or negative, and its magnitude is the norm of TPDV. The position/orientation, which can be obtained by resolving the intersection with the TPDV and the RPDV, is the contact position/orientation. The contact position is described by the center of rotation of the target. The contact normal is the direction which separates the target and the opponent. The end points on the target and the opponent, which are used to calculate the TPDV and the RPDV, are the contact points between the opponent and the target in contact position and orientation. When there are multiple contact points between the target and opponent, only one of the contact point pairs can be obtained.

If the activities of both groups are SC_ACTIVITY_ACTIVE, the penetration depth computation in which the target and opponent are exchanged is also performed and the results about both directions can be obtained.

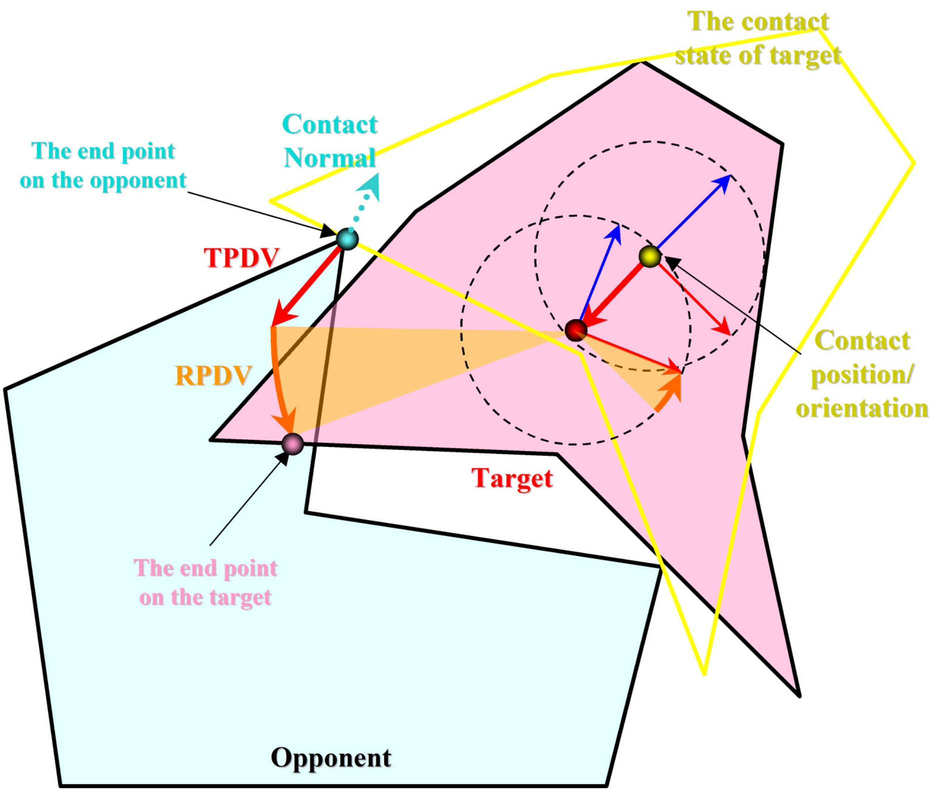


Figure 7-7: Penetration depth computation

The penetration depth computation has parameters, which control how precisely the calculation should be performed, since an exact solution might take too much time for real time applications. SC_SCENE_MANAGER_TOLERANCE specifies the tolerance which controls how much error is permitted for penetration depth computation. SC_SCENE_MANAGER_MAX_ITERATION specifies the maximum iteration at which penetration depth computation terminates, even if the solution has not converged. If the maximum displacement of transformations is around δ_{MAX}, the following relation should be satisfied, ideally.

$$\delta_{MAX} \approx C_S \varepsilon I_{MAX} \qquad (7\text{-}1)$$

Here, ε, C_S, I_{MAX} mean respectively the tolerance of penetration depth computation, the safety coefficient of penetration depth computation and the maximum iteration. The default value of C_S is 0.49 and it should not be changed.

List 7-13 shows how to set the tolerance and maximum iteration of penetration depth computaiton.

List 7-13: How to set the tolerance value and maximum iteration of penetration depth computation

```
SCdouble tolerance=0.1;// the tolerance value of calculation
SCdouble safetyCoefficient=0.49;// the safety coefficient
SCint maxIteration=10; // maximum iteration
SCSceneManager scene(SC_SCENE_MANAGER_CLOSED_POLYHEDRA);
// Setting of transformation and attributes of SCSceneManager
…
scene.SetAttributeDouble(SC_SCENE_MANAGER_TOLERANCE,tolerance);
scene.SetAttributeInteger(SC_SCENE_MANAGER_SAFETY_COEFFICIENT,safetyCoefficient);
scene.SetAttributeInteger(SC_SCENE_MANAGER_MAX_ITERATION,maxIteration);
```

Penetration depth computation differs not only between geometry data type, but also between previous transformations of the object. Figure 7-8 and Figure 7-9 shows penetration depth computation of closed polyhedra (solid model) and triangle soup (surface model) respectively. Even if there is no intersection about the current transformation of objects, it might happen that the objects are penetrating.

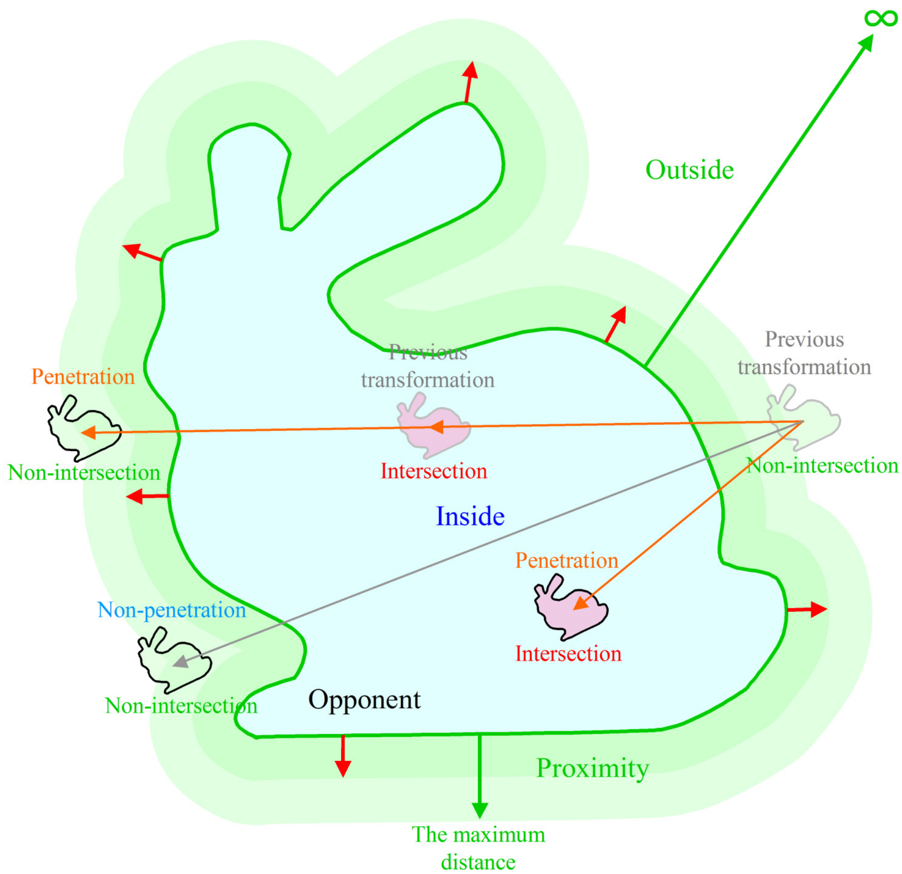


Figure 7-8: Penetration depth computation of closed polyhedra (solid model)

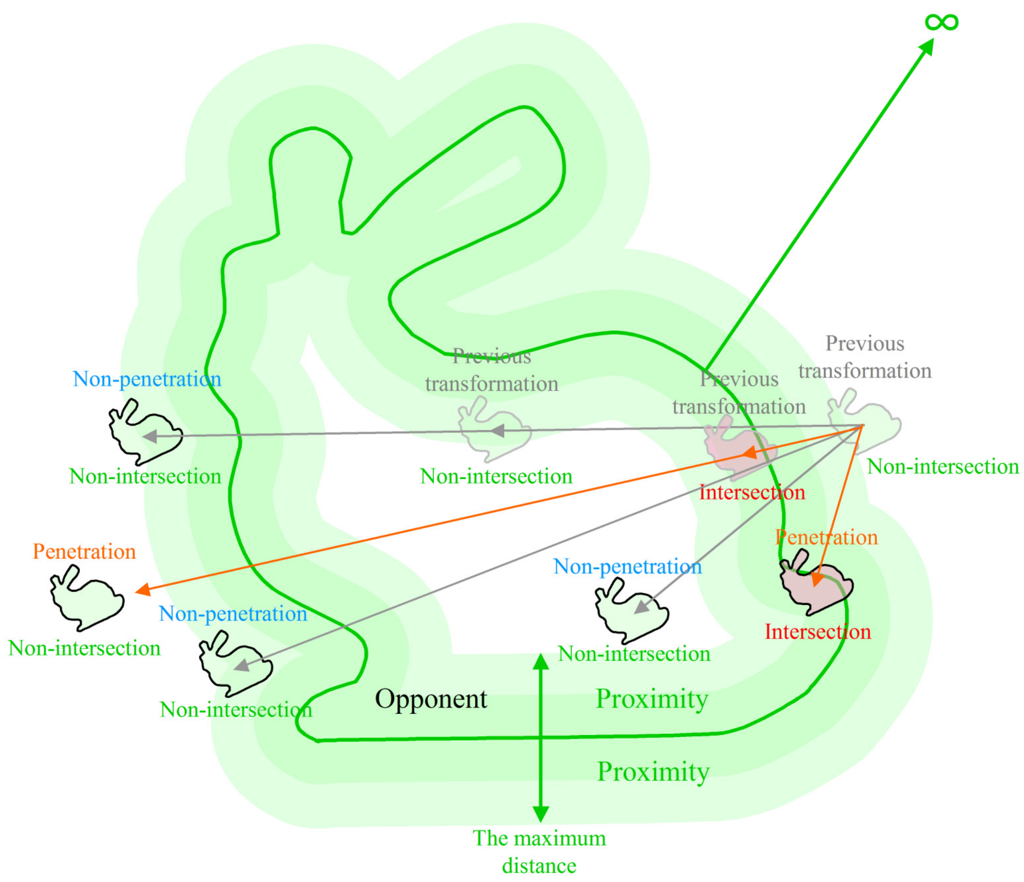


Figure 7-9: Penetration depth computation of triangle soup (surface model)

7.7 Rotation mode

As the combination of TPDV and RPDV cannot be determined uniquely, this SDK provides some rotation modes which determine the specific combination of TPDV and RDPV to return. Table 7-2 show the rotation modes.

Table 7-2: Rotation modes

Rotation mode	Description
None rotation mode	The orientation of contact state of target keeps the value at the time when the penetration happened.(As shown in Figure 7-10 (b))
TPD minimization mode (Input rotation mode)	The combination of the TPDV and the RPDV is determined such that minimization of RPDV takes priority over to TPDV. (As shown in Figure 7-10 (c))
RPD minimization mode (Free rotation mode)	The combination of TPDV and RPDV is determined such that minimization of TPDV takes priority over to minimization of RPDV. (As shown in Figure 7-10 (d))
Potential minimization mode (Mix rotation mode)	The combination of TPDV and RPDV is determined such that potential has the minimum value. (As shown in Figure 7-10 (e))

List 7-14 show how to execute collision detection, according to the current transformation and configuration.

List 7-14: How to set rotation mode

```
SCSceneManager scene(SC_SCENE_MANAGER_CLOSED_POLYHEDRA);
// Setting of transformation and attributes of SCSceneManager
…
scene.SetAttributeEnum(SC_SCENE_MANAGER_ROTATION_MODE,SC_ROTATION_MODE_INPUT);
```

In mix rotation mode, the TPDV and RPDV depend on the ratio of the coefficients of stiffness k_T, k_R in （ 4-5 ）. The coefficients of stiffness can be set as shown in List 7-15.

List 7-15: How to the coefficients of stiffness for potential minimization mode

```
SCdouble tpdvStiffness=0.6; // Coefficient of TPDV for mix rotation mode
SCdouble rpdvStiffness=150;// Coefficient of RPDV for mix rotation mode
SCSceneManager scene(SC_SCENE_MANAGER_CLOSED_POLYHEDRA);
// Setting of transformation and attributes of SCSceneManager
…
scene.SetAttributeEnum(SC_SCENE_MANAGER_FORCE_STIFFNESS,tpdvStiffness);
scene.SetAttributeEnum(SC_SCENE_MANAGER_TORQUE_STIFFNESS,rpdvStiffness);
```

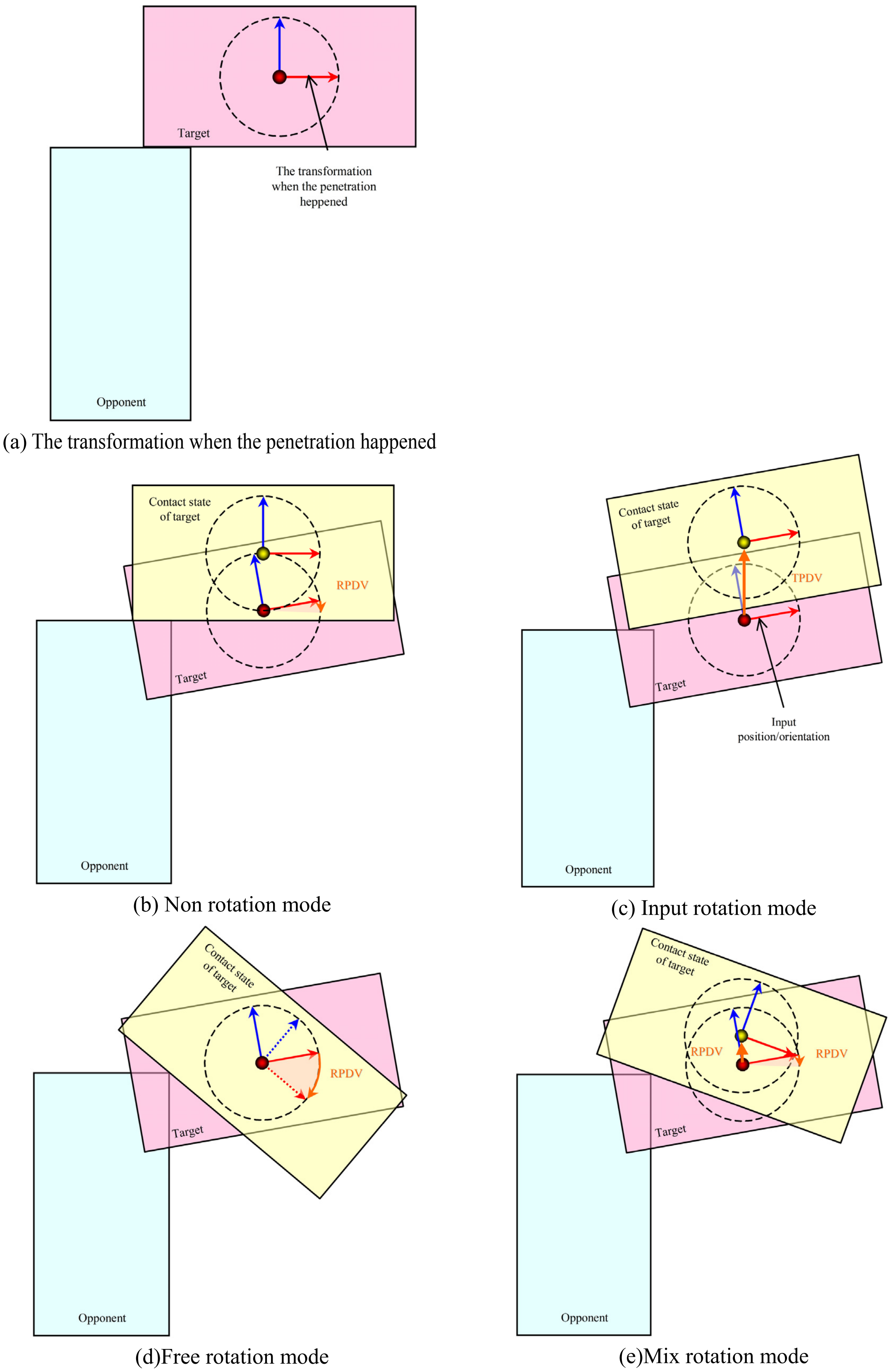


Figure 7-10: Rotation modes

7.8 Execution of Collision Detection

List 7-16 shows how to execute collision detection, according to the current transformation and configuration.

List 7-16: How to execute collision detection

```
SCSceneManager scene(SC_SCENE_MANAGER_CLOSED_POLYHEDRA);
// Setting of transformation and attributes of SCSceneManager
…
scene.UpdateStatus();
```

7.9 Getting results of collision detection

Minimum distance computations and penetration depth computations are performed on each pair of groups according to their activity.

List 7-17 shows how to get the number of pairs.

List 7-17: How to get the number of pairs

```
SCSceneManager scene(SC_SCENE_MANAGER_CLOSED_POLYHEDRA);
// Setting of transformation and attributes of SCSceneManager
// Execution of collision detection of current configurations
…
SCint count;
scene.GetStatus(SC_PAIR_COUNT,&count);//get the number of pairs

for(int i=0;i<count;i++){
    // Get status about each pair
}
```

The statuses of each pair of groups can be retrieved using the methods shown in List 7-19.

List 7-18: Methods to get the statuses of each pair of groups

```
SCint SCSceneManager::GetStatus(SCenum type,SCint*status,SCint index,SCbool reverseFlag);
SCint SCSceneManager::GetStatus(SCenum type,SCfloat*status,SCint index,SCbool reverseFlag);
SCint SCSceneManager::GetStatus(SCenum type,SCdouble*status,SCint index,SCbool reverseFlag);
```

Here, *index* specifies which pair of groups the status information is in relation to. In each pair of groups, there are two results according to the direction. If *reverseFlag* is false, the result is about one of the directions. If *reverseFlag* is true, the result is about the other direction.

List 7-19 shows how to get the status information of each pair.

List 7-19: How to get the status of each pair.

```
for(int i=0;i<count;i++){
    int result1,result2;
    scene.GetStatus(SC_STATUS_RESULT,&result1,i,false);
    scene.GetStatus(SC_STATUS_RESULT,&result2,i,true);
    switch(result1){
    case SC_NO_ERROR:
        // Minimum distance compultation or penetration depth computation has succeeded
        break;
    case SC_ERROR_INVALID_INITIAL_TRANSFORMATION:
        // Penetration depth computation has failed because initial transformation was invalid
        break;
    case SC_ERROR_UNKNOWN_DISTANCE:
        // There is no intersection, but distance is unknown because the distance is greater
than the threshold
        break;
    case SC_ERROR_NO_RESULT:
        // There is no result in this direction
        break;
    }
    switch(result2){
    case SC_NO_ERROR:
        // Minimum distance compultation or penetration depth computation has succeeded
        break;
    case SC_ERROR_INVALID_INITIAL_TRANSFORMATION:
        // Penetration depth computation has failed because initial transformation was invalid
        break;
    case SC_ERROR_UNKNOWN_DISTANCE:
        // There is no intersection, but distance is unknown because the distance is greater
than the threshold
        break;
    case SC_ERROR_NO_RESULT:
        // There is no result in this direction
        break;
    }
}
```

List 7-20 shows how to get the status information (such as group IDs, object IDs, distance, contact normal, end points, TPDV, RPDV, contact position/orientation) of one of the directions (specified by *reverseFlag*) with respect to one of the pairs (specified by *index*). The statuses of the other directions can be obtained by changing *reverseFlag* from false to true.

List 7-20: How to get status information.

```
SCint result;
SCint gids[2];
SCint oids[2];
SCdouble distance;
SCdouble normal[3];
SCdouble point1[3],point2[3];
SCdouble tpdv[3],rpdv[3];
SCdouble contactPosition[3],contactOrientation[4];

scene.GetStatus(SC_GROUP_ID,gids,i,false);// Get the group IDs
scene.GetStatus(SC_STATUS_RESULT,&result,i,false); // Get the result
switch(result){
case SC_NO_ERROR:
    // Minimum distance compultation or penetration depth computation has succeeded
    scene.GetStatus(SC_OBJECT_ID,oids,i,false);// Get the group IDs
    scene.GetStatus(SC_DISTANCE,&distance,i,false);// Get the distance
    scene.GetStatus(SC_CONTACT_NORMAL,normal,i,false);// Get the contact normal
    scene.GetStatus(SC_POINT_ON_TARGET,point1,i,false);// Get the end point on the target
    scene.GetStatus(SC_POINT_ON_OPPONENT,point2,i,false);// Get the end point on the opponent
    if(distance<=0){
        // Penetration depth computation was performed
        scene.GetStatus(SC_TPD_VECOTR,tpdv,i,false);// Get the TPDV
        scene.GetStatus(SC_RPD_VECOTR,rpdv,i,false);// Get the RPDV
        scene.GetStatus(SC_CONTACT_POSITION,
                        contactPosition,i,false);   // Get the contact position
        scene.GetStatus(SC_CONTACT_ORIENTATION,
                        contactOrientation,i,false);// Get the contact orientation
    }else{
        // Minimum distance compultation was performed
    }
    break;
case SC_ERROR_INVALID_INTIAL_TRANSFORMATION:
    // There is intersection, but penetration depth computation could not be performed.
    break;
case SC_ERROR_UNKNOWN_DISTANCE:
        // There is no intersection, but distance is unknown because the distance is greater
than the threshold
    break;
case SC_ERROR_NO_RESULT:
    // There is no information
default:
    // Fatal error
    break;
}
```

8. Standard coding flow

Figure 8-1 shows a standard coding flow for a VR application using SmartCollisionSDK. Usually, in addition to the collision loop, there is a graphics loop, a haptic loop and event handling functions.

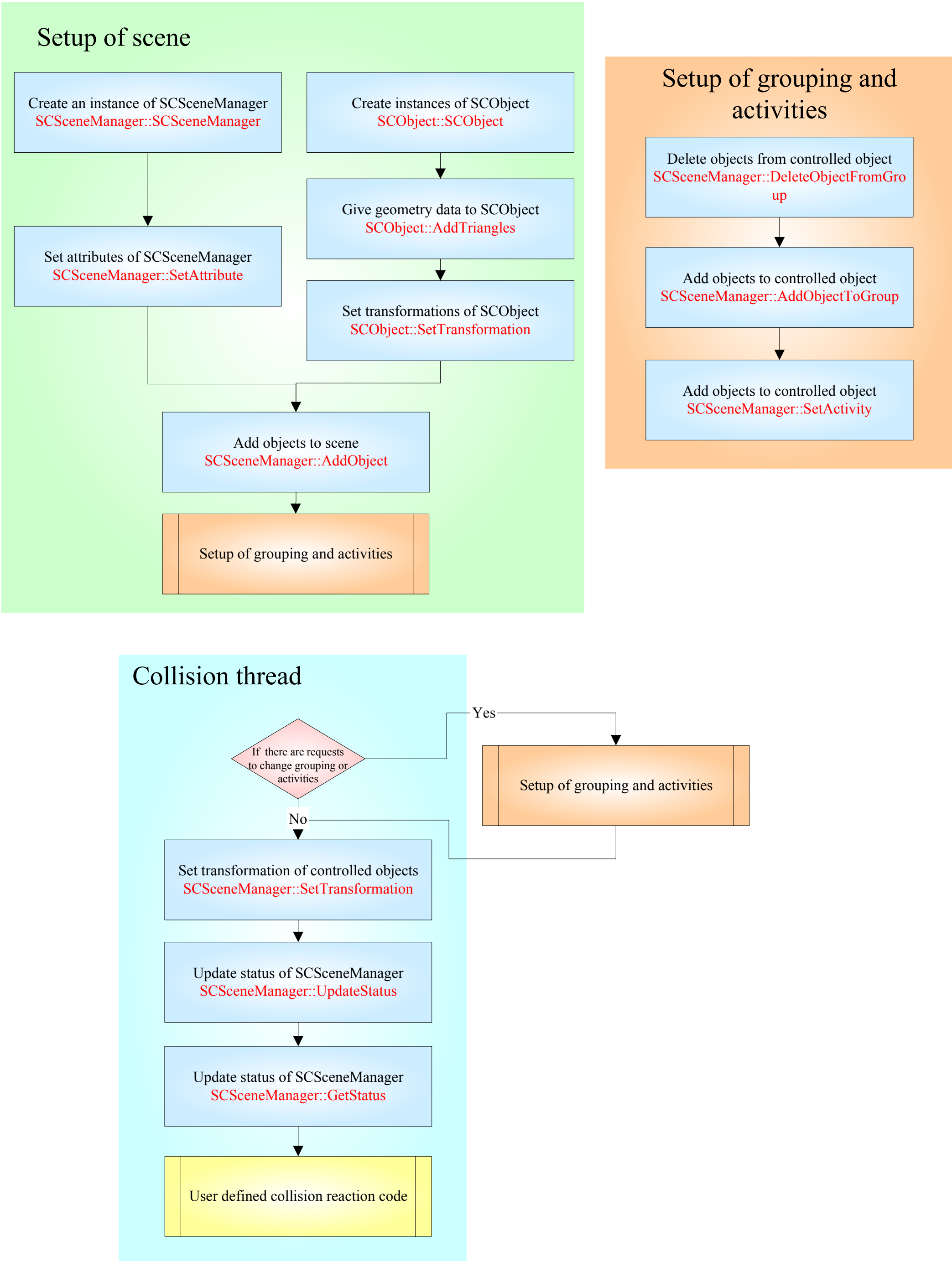


Figure 8-1: Standard coding flow

9. Example programs

9.1 How to build examples

Enter the *example* directory in the SDK package and choose the example you wish to build. Within that directory are files which will automatically set up the correct build environment within Visual Studio. If you are using Visual Studio 6, you should double click on the .DSW file to begin building the examples. If you are using later versions, you should double click on the .SLN file.

The example directory may include at least two such files, one to build an executable with mouse input support, and one to build an executable with support for the Phantom haptics input device.

Next you must choose a solution configuration to build. There should always be configurations for both Debug and Releases, but additionally there may be a configuration to build in other modules such as the Smart Polygon Optimizer (SPO). Before you build such a configuration, be sure to verify that your dongle supports building with those modules, or you may get a runtime error.

The examples included with the SDK only have two external build dependancies: Open Haptics Toolkit and GLUT. However GLUT is often distributed with OHT so if you have OHT installed then you should have GLUT too. Otherwise you will have to find and install GLUT manually.

If the build complains that it cannot find a .H header file or a .LIB dependancy for OHT or GLUT, you may have to manually add them to the project’s properties from the Solution Explorer in the Additional Include Directories and the Additional Dependancies properties respectively.

9.2 HelloSmartCollision!

This sample program is stored below.

SmartCollisionSDK/examples/HelloSmartCollision

List 9-1 shows a simple program of SmartCollisionSDK. As shown in Figure 9-1, there are two tetras in the scene. At first, the controlled objects is in {1,1,1}, and distance is $\sqrt{4/3}$. Second, the controlled object is {0,0,1/2}, and penetration depth is $\sqrt{1/12}$.

List 9-2 shows result of HelloSmartCollision.cpp.

List 9-1: HelloSmartCollision.cpp

```
// HelloSmartCollision.cpp

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

#include "sc.h"

double vertices[12]={0,0,0,
   1,0,0,
   0,1,0,
   0,0,1};
int triangles[12]={0,2,1,
   1,3,0,
   0,3,2,
   1,2,3};

int main(int argc, char* argv[])
{
   SCSceneManager manager(SC_SCENE_MANAGER_CLOSED_POLYHEDRA);
   SCObject object1(SC_OBJECT_TYPE_CLOSED_POLYHEDRA);
   SCObject object2(SC_OBJECT_TYPE_CLOSED_POLYHEDRA);
   double position1[3]={1.0,1.0,1.0};
   double position2[3]={0.0,0.0,0.5};
   double distance;
   // Add geometry to objects
   if(object1.AddTriangles(vertices,4,triangles,4)==SC_ERROR_INVALID_LICENSE){
      // license check error happened
      printf("Error: Invalid lincense.\n");
      system("pause");
      return -1;
   }
   object2.AddTriangles(SC_OBJECT_TYPE_CLOSED_POLYHEDRON,vertices,4,triangles,4);
   // Set attributes of SCSceneManager
   manager.SetAttributeDouble(SC_SCENE_MANAGER_MAX_DISTANCE,10.0);
   manager.SetAttributeDouble(SC_SCENE_MANAGER_TOLERANCE,0.1);
   manager.SetAttributeInteger(SC_SCENE_MANAGER_MAX_ITERATION,10);
   // Add objects SCSceneManager
   manager.AddObject(1000,&object1);
   manager.AddObject(2000,&object2);
   // Add objects to the group with ID 0
   manager.AddObjectToGroup(0,1000);
   // Print a line
   printf("Hello SmartCollision!\n");
   // Set transformation for the group with ID 0
   manager.SetTransformation(SC_POSITION_ORIGIN,position1,0);
   // Execution of collision detection
   manager.UpdateStatus();
   // Get status of collision detection
   manager.GetStatus(SC_DISTANCE,&distance);
```

```
    // Print results
    printf("In position=%lg,%lg,%lg\n",position1[0],position1[1],position1[2]);
    printf("  Distance=%lg(=sqrt(4/3))\n",distance);
    // Set transformation for the group with ID 0
    manager.SetTransformation(SC_POSITION_ORIGIN,position2,0);
    // Execution of collision detection
    manager.UpdateStatus();
    // Get status of collision detection
    manager.GetStatus(SC_DISTANCE,&distance);
    // Print results
    printf("In position=%lg,%lg,%lg\n",position2[0],position2[1],position2[2]);
    printf("  Pentration depth=%lg(=sqrt(1/12))\n",-distance);
    system("pause");
    return 0;
}
```

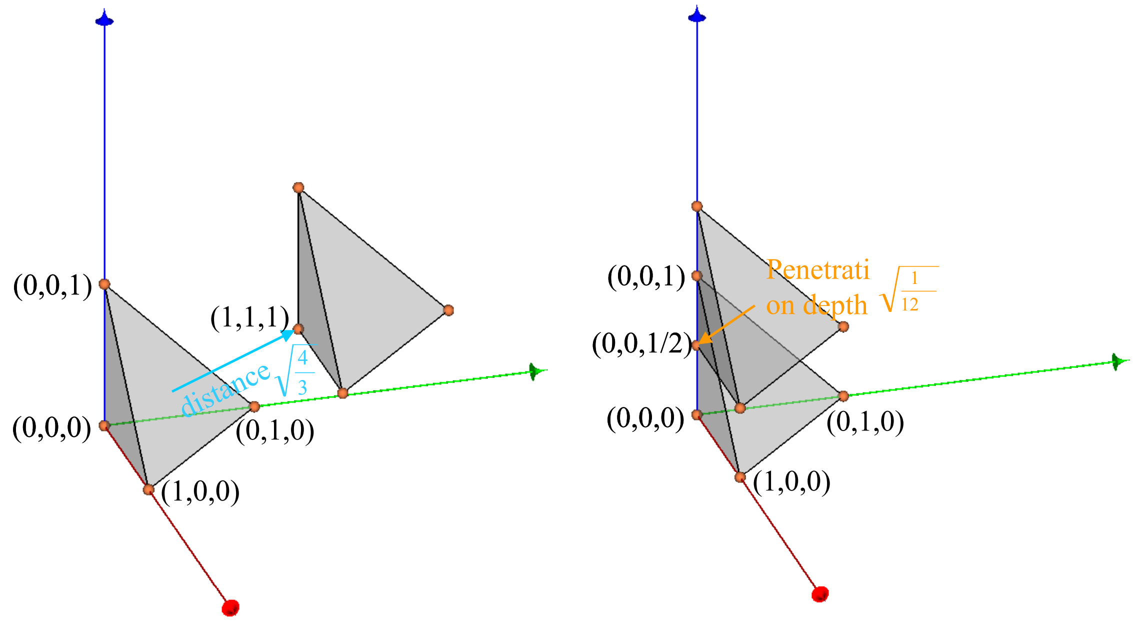


Figure 9-1: Sample program's scene

List 9-2: Result of HelloSmartCollision.cpp

```
Hello SmartCollision!
In position=1,1,1
  Distance=1.1547(=sqrt(4/3))
In position=0,0,0.5
  Pentration depth=0.288675(=sqrt(1/12))
```

9.3 SimpleCollisionTest

This sample program is stored below.

SmartCollisionSDK/examples/SimpleCollisionTest

This program is also a simple of SmartCollisionSDK. This program loads two objects from files and moves one of the object and get collision information at each step.

9.4 SmartCollisionTest

This sample program is stored below.

SmartCollisionSDK/examples/SmartCollisionTest

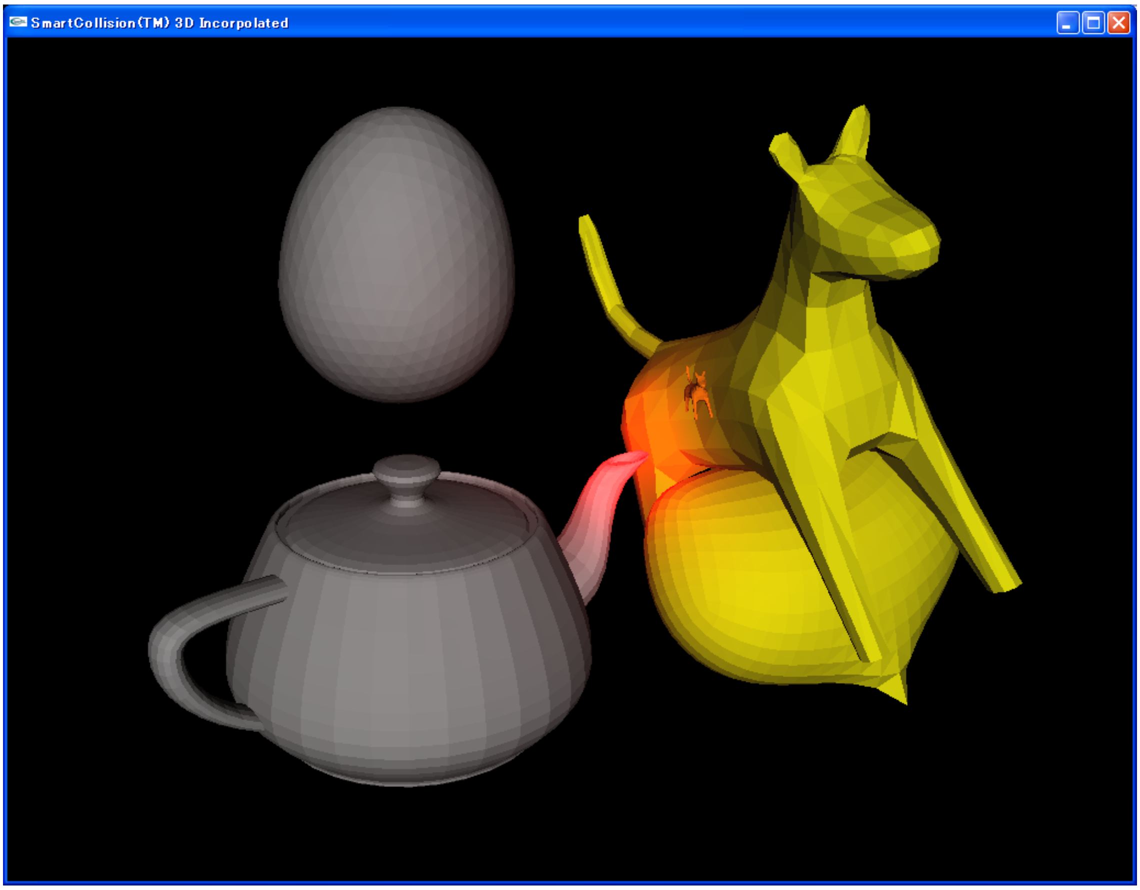


Figure 9-2: A screenshot of SmartCollisionTest

This example shows how to manage collision detection between static group and another group consisting multiple objects.

This example also shows how to integrate user I/O device (a general mouse and a SensAble PHANTOM haptic device) and SmartCollisionSDK. This example requires GLUT 3.7(The OpenGL Utility Toolkit)

http: //www.opengl.org/resources/libraries/glut/glut_downloads.html

For Phantom:

SmartCollisionTestPhantom.sln is a solution file for VC.NET 2003 or later.

SmartCollisionTestPhantom.vcproj(for VC .NET 2003)

libEasyDevicePhantom.vcproj(for VC .NET 2003)

SmartCollisionTestPhantom.dsw is a work space file for VC6.

SmartCollsionMultipleGroupPhantom.dsp(for VC6)

libEasyDevicePhantom.dsp(for VC6)

SensAble PHANTOM device and Open Haptic Toolkit 1.00 or later are required.

For Mouse:

SmartCollisionTestPhantom.sln is a solution file for VC.NET 2003 or later.

SmartCollisionTestMouse.vcproj(for VC .NET 2003)

libEasyDeviceMouse.vcproj(for VC .NET 2003)

SmartCollisionTestPhantom.dsw is a work space file for VC6.

SmartCollisionTestMouse.dsp(for VC6)

libEasyDeviceMouse.dsp(for VC6)

EasyDevice class is a common interface class for mouse and PHANTOM. The implementations for each device (EPhantom and EMouse) are defined as implementation class of EasyDeviece. Other most part of the source code is common.

If you'd like to support other devices like a 3D mouse or magnetic sensor or so on, you can define additional implementation class for the device. See source code comments for details.

9.5 SmartCollisionMultipleGroupTest

This sample program is stored below.

SmartCollisionSDK/examples/SmartCollisionMultipleGroupTest

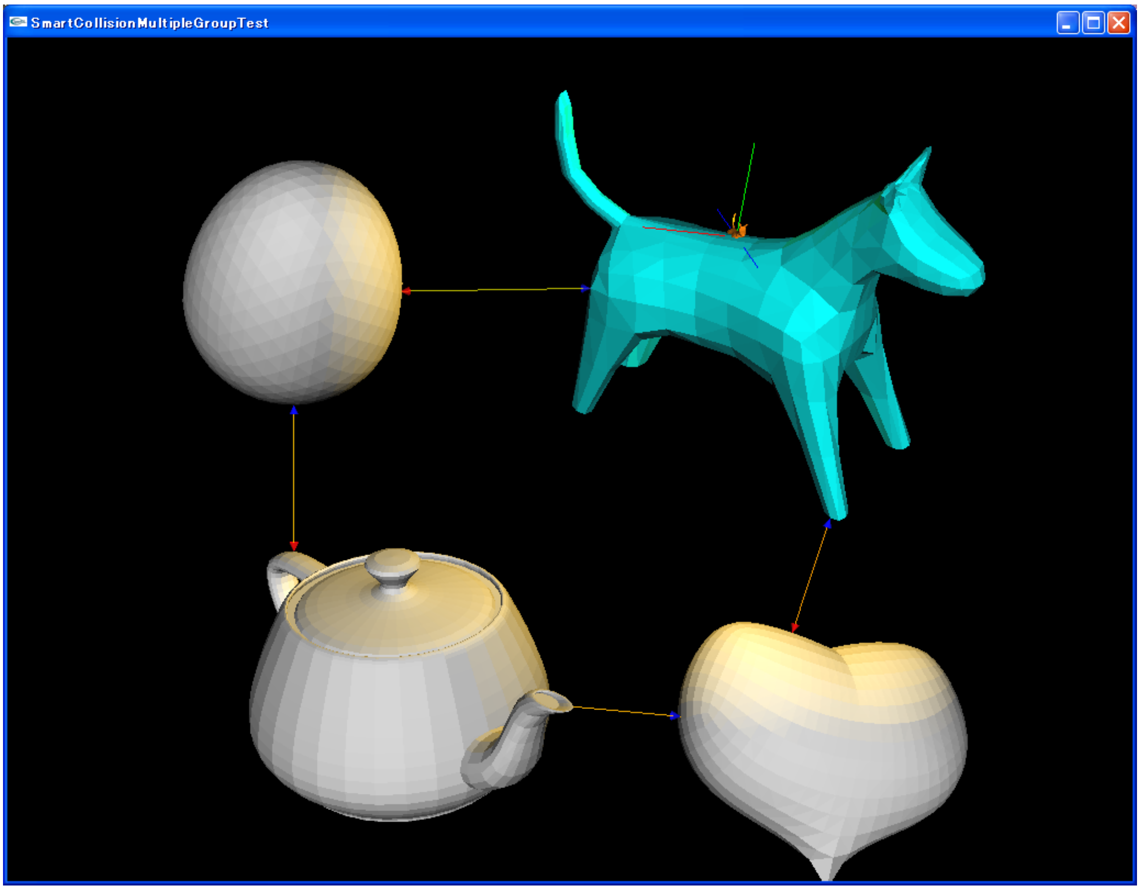


Figure 9-3: A screenshot of SmartCollisionMultipleGroupTest

This example shows how to manage collision detection between multiple groups.

This example also shows how to integrate user I/O device (a general mouse and a SensAble PHANTOM haptic device) and SmartCollisionSDK. This example requires GLUT 3.7(The OpenGL Utility Toolkit)

[http: //www.opengl.org/resources/libraries/glut/glut_downloads.html](http://www.opengl.org/resources/libraries/glut/glut_downloads.html)

For Phantom:

SmartCollisionMultipleGroupTestPhantom.sln is a solution file for VC.NET 2003 or later.

SmartCollisionMultipleGroupPhantom.vcproj(for VC .NET 2003)

libEasyDevicePhantom.vcproj(for VC .NET 2003)

SmartCollisionMultipleGroupTestPhantom.dsw is a work space file for VC6.

SmartCollsionMultipleGroupPhantom.dsp(for VC6)

libEasyDevicePhantom.dsp(for VC6)

SensAble PHANTOM device and Open Haptic Toolkit 1.00 or later are required.

For Mouse:

SmartCollisionMultipleGroupTestPhantom.sln is a solution file for VC.NET 2003 or later.

SmartCollisionMultipleGroupMouse.vcproj(for VC .NET 2003)

libEasyDeviceMouse.vcproj(for VC .NET 2003)

SmartCollisionMultipleGroupTestPhantom.dsw is a work space file for VC6.

SmartCollsionMultipleGroupMouse.dsp(for VC6)

libEasyDeviceMouse.dsp(for VC6)

EasyDevice class is a common interface class for mouse and PHANTOM. The implementations for each device (EPhantom and EMouse) are defined as implementation class of EasyDeviece. Other most part of the source code is common.

If you'd like to support other devices like a 3D mouse or magnetic sensor or so on, you can define additional implementation class for the device. See source code comments for details.