Introduction to Operators in CAD Systems

All CAD systems provide some functionality to the user, to simplify the process of completing the design. We will look at some of the basic operators that are provided to the user. In particular cases, we will attempt to get a better understanding of the mathematical foundations behind these operators. We have two objectives:
(a) Which operators are available: to help us to plan, step by step, how to complete the design of a part.
(b) The understanding of each operator: to help us to decide when to use an operator, when not to use it, and, if the application of an operator fails, to have some insight on why it failed. This knowledge can often help us to reset some parameters and successfully complete the design.

Basic operators:

1. Transformations

Building and viewing any 3D model in the computer basically involves two types of affine transformations – object transformations and coordinate transformations. The underlying mathematics for each of these is remarkably similar –we can use matrices to work with all such transformations.
Affine transformations include translations (an entity is moved along a given straight line by some distance) or rotations (about a given axis, by a given angle).

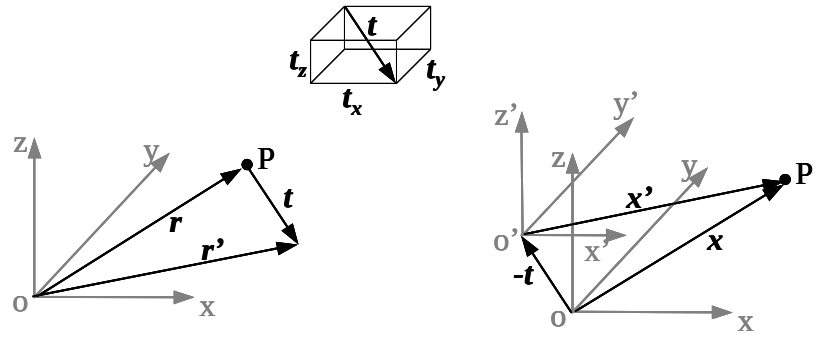
All geometric entities we store in a CAD system (line, curve, face etc.) are described in terms of the coordinates of their vertices. Hence, transformation of the entity can be done by transforming each vertex of the entity. In other words, the shape/size of our entities is *invariant with respect to translation and rotation*. Hence we need only study translation, rotation of a point (vertex). We shall use matrix notation, and denote points by column vectors: $[x, y, z]^T$.

Translation:
If we want to move a point $\boldsymbol{r}$ $[x, y, z]^T$, by $[t_x, t_y, t_z]^T$ units in the X-, Y- and Z-directions respectively with respect to a fixed (global) coordinate frame, then its new coordinates, $\boldsymbol{r'}$ can be obtained as:

$\boldsymbol{r'} = \boldsymbol{r} + \boldsymbol{t}$,
where $\boldsymbol{t}$ is the row vector $[\ t_x, t_y, t_z]^T$

This is equivalent to saying that we stand at the origin of a moving coordinate frame, and that our coordinate frame moves by $\boldsymbol{-t}$ with respect to the global coordinate frame in which the point $\boldsymbol{P}$ is fixed. Let us denote the position of point P in the fixed frame, *oxyz*, by $\boldsymbol{x}$. Then translating the moving coordinate frame by –t gives us the frame *o'x'y'z'*. The position of the point P in the new coordinate frame is $\boldsymbol{x'}$. Then, as seen in the figure below, the vectors $\boldsymbol{r'}$ and $\boldsymbol{x'}$ are identical.

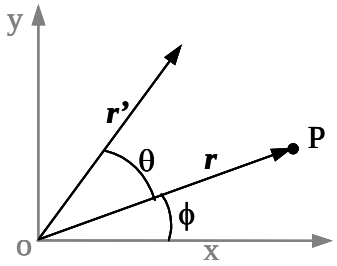
In other words, an object translation by $\boldsymbol{t}$ is identical to a coordinate translation by $\boldsymbol{-t}$.



Rotation:

Let us look at rotations in the XY plane first (i.e. rotation of a point P by angle θ about the *oz* axis). Let us call the matrix *A* as *Rot(z,θ)*.

$$\boldsymbol{r} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos\phi \\ r\sin\phi \end{bmatrix}$$



$$\boldsymbol{r}' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} r\cos(\phi+\theta) \\ r\sin(\phi+\theta) \end{bmatrix}$$

$$= \begin{bmatrix} r\cos\theta\cos\phi - r\sin\theta\sin\phi \\ r\sin\theta\cos\phi + r\cos\theta\sin\phi \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} r\cos\phi \\ r\sin\phi \end{bmatrix} = A\begin{bmatrix} x \\ y \end{bmatrix} = A\boldsymbol{r}$$

Similarly, consider a point, ***x***, fixed in the coordinate frame *oxyz*. If we compute its coordinates, ***x'***, in a new coordinate frame, *ox'y'z'* that is obtained by rotating *oxyz* by angle $-\theta$ around the *oz* axis, then we find that ***x'*** is the same as ***r'***. In other words, an object transformation by Rot(z, θ) is the same as a coordinate transformation by Rot(z, - θ).
Notice that rotation about the oz axis does not change the z-coordinate of a point. Therefore, it is easy to extend this transformation to 3D points (you can verify it by doing the matrix multiplication).

$$\boldsymbol{r}' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = Rot(z,\theta)\boldsymbol{r}$$

And similarly, we can derive the expressions for Rot(x, θ) and Rot(y, θ):

$$\boldsymbol{r}' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = Rot(y,\theta)\boldsymbol{r}$$

$$\mathbf{r}' = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = Rot(x,\theta)\mathbf{r}$$

Two other formulas of rotation are useful in CAD, and you can read about them in the advanced notes on transformations. I will also discuss why it is useful to use the matrix notation in those notes.

Applications:
(a) Maintaining the design feature tree: Most CAD systems maintain the sequence of operations used to create a part design in a design feature tree, which is similar to a CSG tree. You define entities in a sketch; each sketch is in its own coordinate frame; then you create a solid shape based on the sketch, and transform the shapes to the correct location before doing a Boolean operation.
(b) Another common application of transformations is in the operators provided for viewing the part. This includes rotate, pan, zoom-in zoom-out etc. In each of these operations, you are basically changing the view-point coordinate frame with respect to the part coordinate frame.

2. Scaling, Non-uniform scaling

Scaling is the application of a scale factor, to shrink or expand the size of a designed part. Uniform scaling can be performed by simply multiplying each coordinate in the BREP of the part by the scaling factor. This has the obvious use in CAD systems that it allows us to design part that have the same geometry but different sizes. for example, if you design a part, but after building the prototype, decide that it should be a little larger/smaller, it is easy to scale the part.

Operation: each vertex, *v(x, y, z)* is transformed to *v'(sx, sy, sz)*, where s is the scaling factor.

Property: Uniform scaling does not change the topology of the part (this is intuitively obvious, since you design a part in arbitrarily selected units, like mm, or cm, or inches – but the computer uses only the numerical values, not the units, in the model).

Non-Uniform scaling is useful in several applications. Typically, this operation involves scaling each vertex coordinate, *v(x, y, z)* by different scaling factors, (s_x, s_y, s_z), so that the transformed coordinates are: $v'(s_x x, s_y y, s_z z)$.

[Question: is it possible to change the topology of the part by non-uniform scaling?]

Applications of Non-uniform scaling:
(a) It is commonly used to modify the mold design for a given part to allow for shrinkage [Question: why non-uniform?]

(b) Clothing and Footwear design: In footwear design, one sample size (e.g. Ladies shoe of size 7) is first designed and tested. If the design is approved, it is scaled to provide the entire range of sizes for different ladies. If size 7, wide shoe is required, the design is scaled only

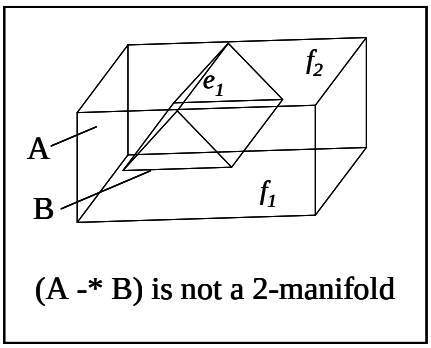
across the width. Likewise, when scaling to get, say, size 5, the scaling factor used for the length is different than the scaling factor for the width.
[Question: Why do we use non-uniform scaling here?]

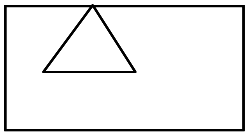
Composition of Transformations

Notice that we have used vectors and matrices to represent different types of transformations – translations, rotations and even scaling. One reason for this is that by some simple extensions, we can use a uniform matrix representation for each transformation. Further, the effect of a series of transformations applied to a geometric entity can be easily computed by just repeatedly the transformed point by the corresponding transformation matrix. This uniform representation is especially useful to CAD programmers, since it simplifies programming and debugging. It also makes it easy to write programs that maintain the sequence of transformations, in case the user would like to *undo/redo* some changes. The exact mechanics of how all this works is discussed in the additional notes on transformations.

3. Boolean operations

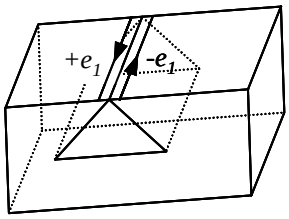
[Regularized] Boolean operations are at the heart of all CAD systems. They include: Union, Intersections, and Difference, and work in the same way as explained for CSG. The exact implementation of a Boolean operation is fairly complex, and will be avoided here. Boolean operators take regular solids (2-manifold) as input, and return 2-manifold solids if the output is a non-empty set. This is complicated since sometimes the result of a Boolean operation on 2-manifold objects is a non 2-manifold object. The figure below shows a Boolean difference operation, Block (A) -* Prism (B).



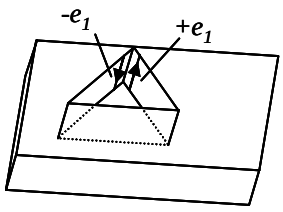


f_1: One loop or two ?

f_2: One loop or two ?



e_1: Two co-edges ? Three ? Four ?

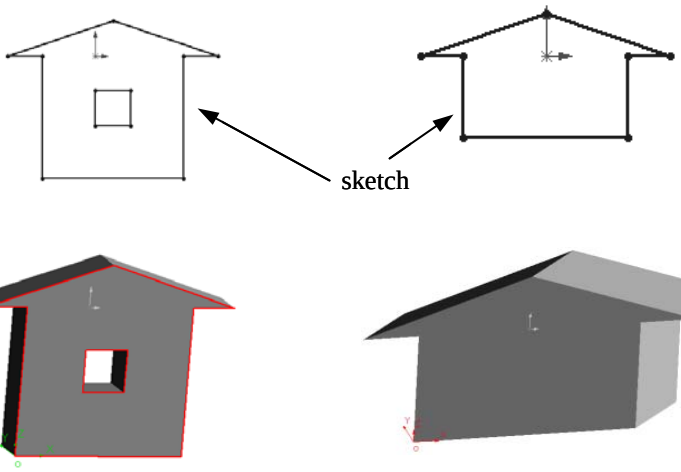


It is important to understand why this is a problem. As you can see from the figure, our version of the winged edge data structure has trouble in storing the data for the object shown above. For instance, we require each edge to be linked to be linked to two co-edges, one positive and one negative. However, for e_1 in the figure, we seem to require two co-edges *of each sign*, four in all. Hence all algorithms based on our data structure will fail. In most modern CAD systems, whenever this type of problem arises, the operation will fail, and you will get an error message (if you are lucky; sometimes the system may just hang up, or crash).
The main point is – many operations in CAD systems are implemented by the use (internally) of Boolean operators. Whenever you try to use such operators, and the resulting object during the calculations is a non 2-manifold, then the system will give you an error message. Having this knowledge, you should be able to re-design your operation so that it can successfully generate the shape you need.

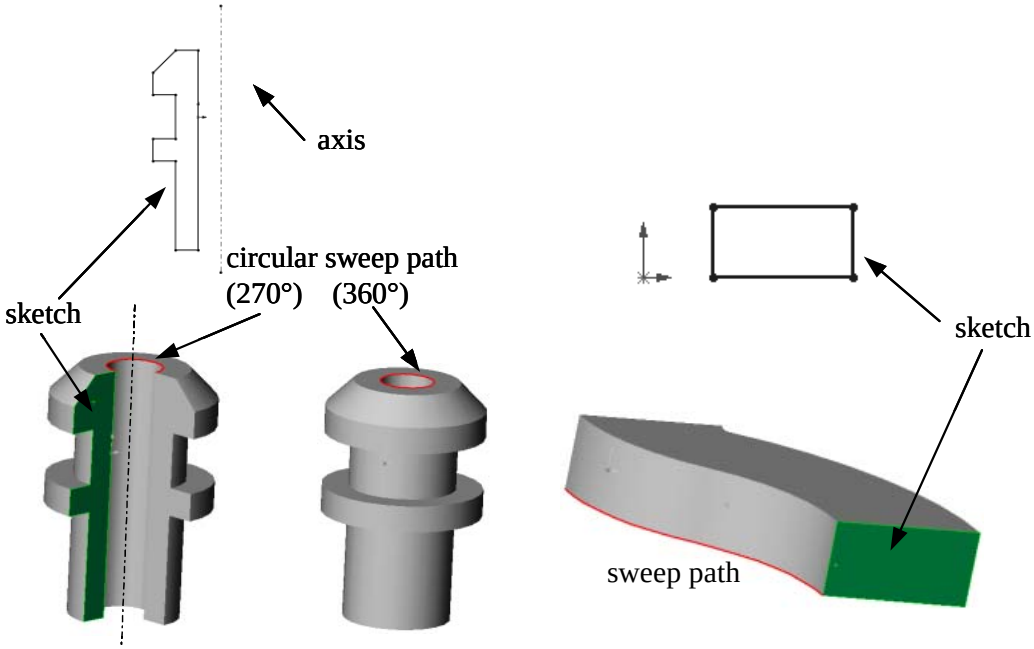
Exercise: Give an example of Regularized Boolean Union where the two inputs are 2-manifolds, but the output is not a 2-manifold object. Try to do the same for regularized Boolean intersection.

4. Sweeping: linear and non-linear

Sweeping is the most popular way for CAD systems to generate solid shapes. Two things need to be defined for a sweep operations: A profile, usually in the form of a 2D sketch, and The sweep path, usually in the form of a continuous, bounded curve. Typical requirements of a swept sketch are: must be made up of one or more loops; no edge on any loop must intersect with any other edge, except at the vertex. Typical requirements of the sweeping path: it should be a single continuous, open or closed curve. The success or failure of such operators depends on the condition that the swept shape *must not intersect with itself* as it moves along the path. You may think of sweeping as follows:
Imagine a coordinate frame in which the sketch is fixed. The origin of this coordinate frame travels along the swept path; at each point on the sweep path, the orientation of the frame with respect to the tangent to the path remains constant; for example, in figure (b) below, the coordinate frame of the sketch always moves such that the swept path is aligned with a vector parallel to [1,1,0]. When the swept path is non-linear and non-planar the frame turns in accordance with tangent vector, as well as the torsion of the curve. All points in 3D space that will be ‘inside’ the swept shape in any position so obtained will be part of the swept volume. The sweeping operation will fail if there is any point that is inside two distinct positions of the swept shape.
It is popular since we have good tools to define 2D shapes – especially sketching tools. The figures below show several different cases (applications) of sweeping based on the sweep path.



(a) Linear sweep normal to sketch (b) Linear sweep oblique to sketch



(c) Circular sweep (d) Complex sweep path

The sweep operator defines a mechanism to obtain 3D volumes from 2D shapes. In this context, figure (a) bears special notice – internally, the system may compute it by doing the following operations in sequence: First sweep the outer loop to get a solid house; then sweep the inner loop to get a rectangular block; Finally, perform a Boolean difference to obtain the house with the hole.

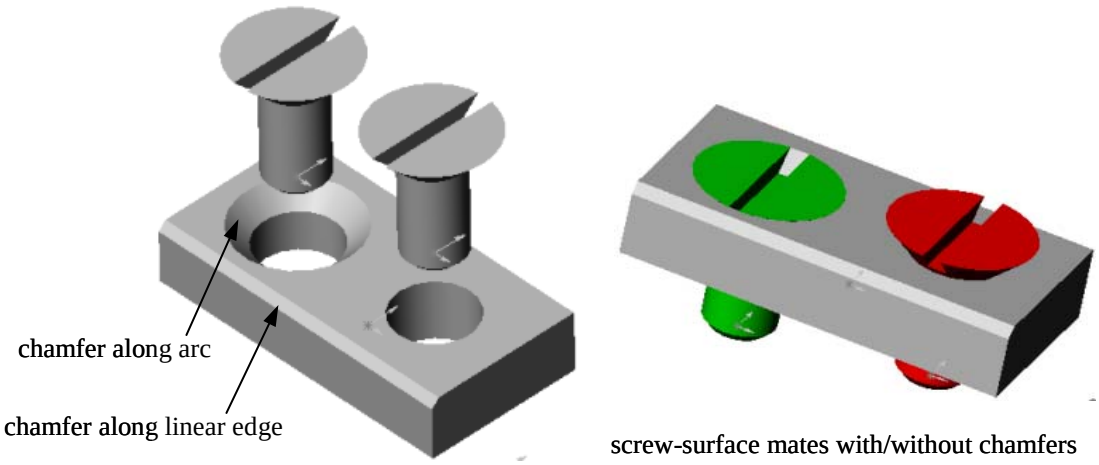
Note: the special case where the sweep curve is a straight line segment perpendicular to the plane of the sketch is often called an ***Extrude operation***. If the extrude operation is used to generate a shape that will be removed from an existing 3D solid, then it is called an ***Extrude-Cut operation***. For instance, the outer house shape in figure (a) can be generated by an extrude operation using the outer loop of the sketch; the hole in the house shape is the result of an extrude-cut operation on the inner loop.

A point that is related to sketching tools used in major CAD systems is that they allow you to define the geometry that is heavily under-dimensioned. Of course, to maintain the solid model, every aspect of the geometry must be well defined; therefore it is clear that internally, these systems make 'reasonable assumptions' about the missing dimensions. Most lazy designers will take full advantage of this convenience and under-dimension their sketches. However, in such models, if a dimensions or a constraint is added later, its effect may be sometimes unexpected. Internally, the CAD system maintains all dimensions, with some information about which dimensions were either explicitly specified by the user either by dimensioning or through setting some constraint, and which dimensions were inferred by the modeler. When adds more dimensions (or constraints), then the system will adjust the geometry by first attempting to adjust one or more of the assumed values. Some other systems allow the user to specify the dimension of some entities in terms of an equation based on other dimensions on the part. Such systems are called parametric CAD systems (e.g. Pro/Engineer from Parametric Technologies); others allow relations between different dimensions to be written as implicit equations; when either value is fixed, the other is computed accordingly. Such systems have been called variational modelers (e.g. I-DEAS from SDRC Inc, which is now owned by EDS).

5.1. Chamfers, Fillets and Rounds

Blending is the name given to an operation that generates a surface, B, between two given surfaces, S1 and S2, such that the interfaces between B and S1, S2 are smooth. The word smooth here may carry different meaning depending on the nature of S1, S2 and sometimes B. Often, smoothness requires the first derivatives to be continuous. Surface blending is a mathematically complex problem. Even simple problems in blending of curves can be fairly difficult, as we shall see in the special case below. A somewhat related operation is that of chamfering. We look at the different classes of blending operations in sequence.

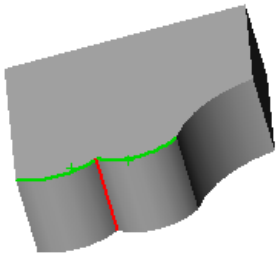
Chamfers: chamfers are angular faces that blunt the sharp edge formed by two faces meeting at a sharp angle, e.g. right- or acute-angled edges. Another use of chamfers is to provide an angled recess at the end of a hole; typically this is used when we don't want a screw passing through the hole to protrude out above the planar face. Since chamfers are planar, therefore they meet the blending surfaces at a sharp (obtuse angled) edge – you can think of them as C^0 blends. The figure below shows examples.



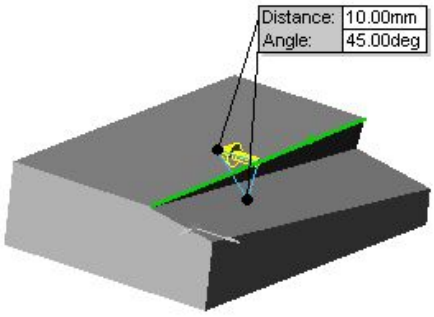
screw-surface mates with/without chamfers

Chamfers such as the ones above are relatively easier to compute – if the edge is linear, the chamfer is a plane and the resulting part geometry is obtained by intersection of this plane with the two faces neighboring the chamfered edge. In the general case, the edge may be a complex curve (usually curves that are neither lines nor conic sections are defined by means of special forms of polynomial parametric functions, called NURBS or B-Splines) the chamfer surface is a ruled-surface (a surface formed by moving a straight line along a (twisting) 3D curve.
Implementing a chamfer feature is therefore not easy. Conceptually, creating a chamfer may be handled by the CAD system as follows:
(a) For each edge that needs to be chamfered, create the chamfer cross-section. This is fairly tricky, since the neighboring faces of this edge may be curved. The chamfer section must be made of two curves in a plane, along these faces, plus a straight line at the chamfer angle, forming a 'triangle'.
(b) Sweep the section of each edge along the corresponding edge to get a solid shape
(c) If two consecutive edges meet at concave faces, the two neighboring chamfer shapes have a 'gap', for which a 'filler shape' must be computed by some (case based) geometric reasoning.
(d) Perform the Boolean operations (Part -* solid shape) for each solid shape and filler shape.

Chamfer operations typically fail if there are complex concavely connected faces, or if the edge is formed by intersection of complex non-planar shapes – i.e. when there are problems during steps (a) of step (c) above. They also fail if, during step (d), the part will become non 2-manifold.
Another case where chamfers fail is if the chamfer feature is so large that it will completely cut out one of the two neighboring faces. Examples of cases where chamfering may fail are shown below.

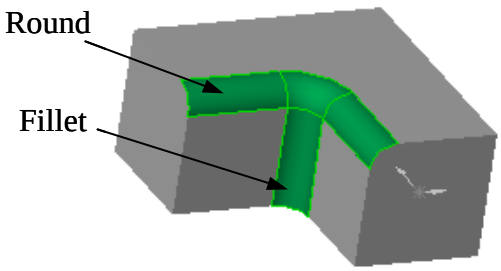


Concavely connected curved face (red edge) is not C^1, difficult to 'merge' the two chamfers corresponding to the two green edges



Green edge difficult to chamfer, since chamfer geometry will destroy the triangular dark face.

Fillets and Rounds are features similar to chamfers, in the sense that their cross-section is composed of a three-edge loop. Two of the edges are the cross-sections of the adjacent faces whose shared edge is used to define the fillet/round. The third edge is a circular arc (in a chamfer, this edge is a straight line). Fillets and rounds have similar shapes; a fillet is a shape that is added to the part along a concave edge; a round is a shape that is removed from the part along a convex edge. The figure below shows examples.



In some CAD systems, both of these features may be called by the same name (fillet). Fillets are useful in design: sharp corners in mechanical parts are usually stress accumulation zones, but providing a fillet can relieve such stress peaks into a uniformly distributed, larger area. Hence all concave edges are filleted in load-carrying mechanical parts. Rounds are useful in manufacturing, especially parts that are moulded. It is difficult to machine a die cavity that has sharp corners, hence all concave corners of the cavity are rounded – and this yields a round in the corresponding (concave) edges of the part. Of course, rounds make the part safer to handle, and are often aesthetically more pleasing, too.

Computation of rounds and fillets by a CAD system follows the same steps as those mentioned for chamfers. Hence this operation will fail in under similar circumstances as for chamfers. Typically, the mathematics and the programming involved is fairly complex for complex shapes. To give you an idea of how complex the geometry can get, here is a simple exercise.

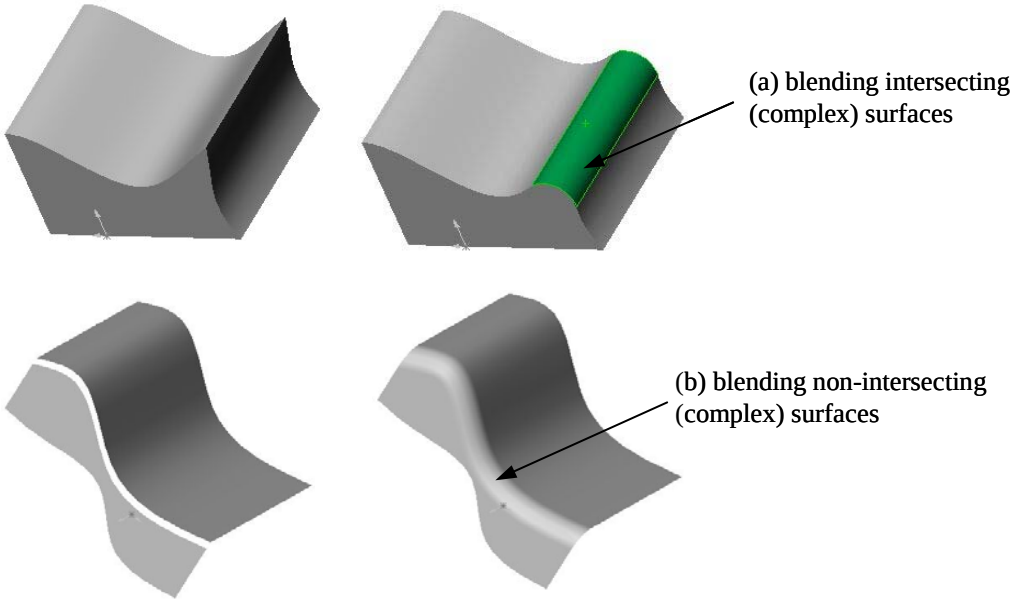
Exercise: ***Filleting a pair of Edges***
Compute the formula for the fillet of radius *r* between two edges in each of the following cases: (a) Two line segments (b) line segment and circular arc, and (c) Two circular arcs.
n each case, the fillet is a circular arc, and your answer must specify the start point, end point and center of the fillet arc.

Chamfers, fillets and rounds are *secondary features* commonly found on most mechanical parts – that is, you cannot design such a feature in isolation. It must refer to another entity, usually an edge on a part.

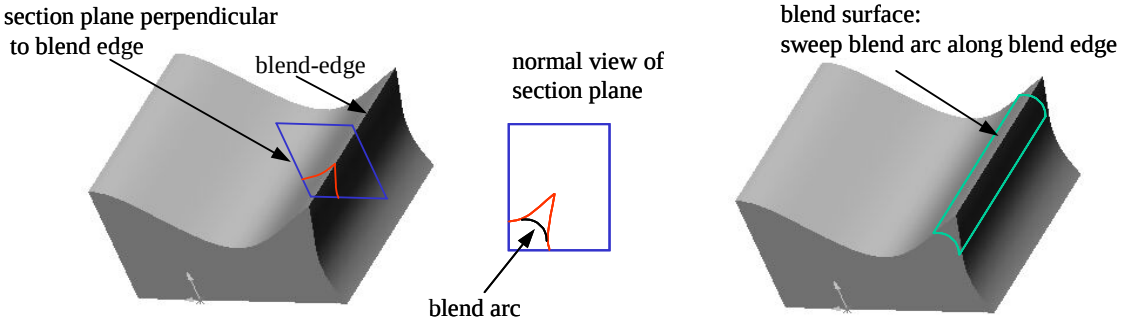
5.2. Blending

Fillets, rounds and chamfers are, in a sense, special cases of the operation called blending. A blend generates a surface that [smoothly] joins two selected surfaces. This is certainly meaningful when the two surfaces intersect along a curve. If the two (bounded) surfaces do not intersect, then the blend is formed using a surface that [smoothly] connects a selected edge of the first surface with a selected edge on the second. Of course, ‘smooth’ here may mean any degree, such as C^0, C^1, etc., but beyond C^1 (and even C^1) is quite difficult to achieve due to other *constraints* we may wish to place upon the blending surface.
In theory, we could define blends in any dimension, and between different entity types (see below), but the main use of blends in mechanical CAD is for smooth merging of surfaces. Also, as the degree of continuity increases (for example, if we want the C^2 continuity), the computations often become inaccurate or complicated. In some applications, for example, designing of body panels of cars, such a high level of continuity is desirable for aesthetic reasons. For a smooth surface, with light reflections in it, the human eye can easily detect a rapid change on curvature (curvature is related to the second derivative).

The mathematics of generalized blending is fairly complex, and we will not go into it here. It is undoubted that blending is a fairly powerful and useful feature provided by most geometry kernels. Typically, it is provided as functions such as filleting or rounding. Another form in which it is provided is as a blend between two complex surfaces. Examples of this function are shown in the figures below, using constant radius blends between two surfaces that are (a) intersecting and (b) non-intersecting.



In the examples above, the blend for case (a) is much simpler to compute, since the cross-section of the blend remains constant. A possible procedure is to take a plane perpendicular to the blending edge to make the cross-section, fit the blending circular arc, and sweep it along the edge to form the blend surface. Intersections of the swept surface with the part surfaces can then be used to compute the resulting part shape.



Computing the blend in case (b) is much more difficult, and we cannot study the mathematics of such operations here. Intuitively, you can imagine the shape generated as follows: imagine a sphere of radius equal to the blend radius, rolling such that it touches both the surfaces at all times. At any position, one point on the sphere touches each surface, and we can draw the great arc on the surface of the sphere joining these points. If we collect all such arcs, they form a continuous surface, which is our required blend. This is called the rolling-ball blend.

When does blending fail ?

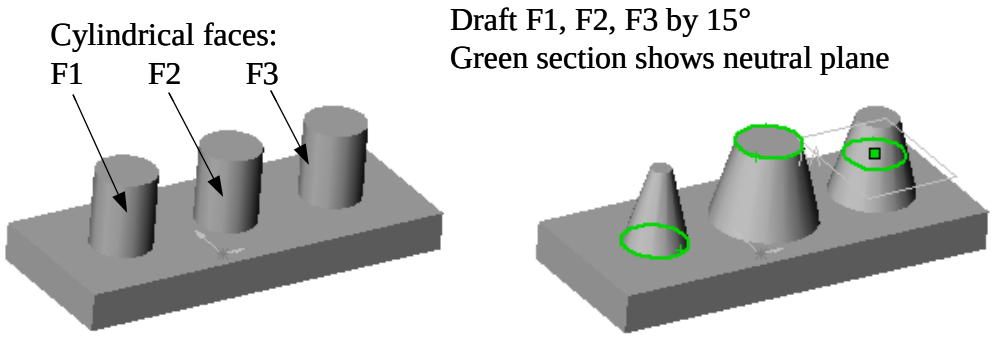
From the description above, it is clear that if at some stage, the distance between the two surfaces is so large that the ball will “fall-through” the gap – then the blend cannot remain continuous; the operation will fail. Another case is if the rolling sphere, at some stage, touches three or more points on the surfaces, simultaneously (this happens if the surface curves sharply). Again, the blend will fail.

6. Tapering and Drafts

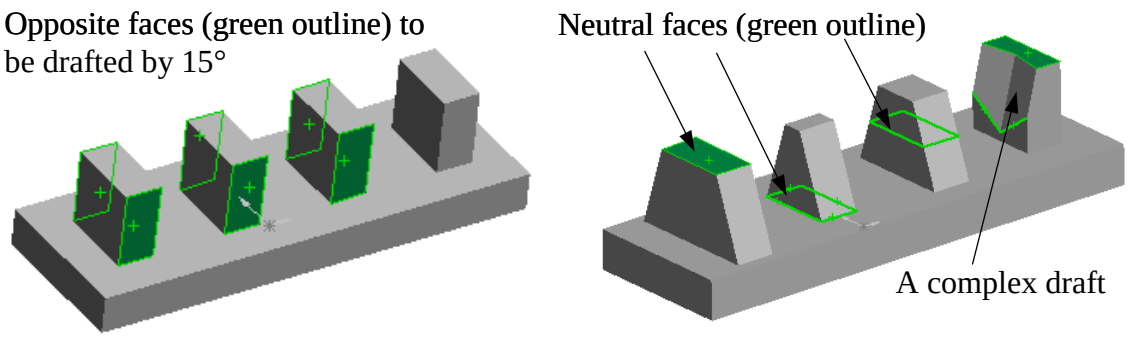
Drafts are typically manufacturing related features, with special importance in parts made by casting or molding. We all know that it is easier to remove a conical part fitted into a cone shaped cavity, rather than a cylinder from a cylindrical hole. This is because (a) the conical part faces no friction as soon as it is moved by a small distance out, and (b) imperfections in the cylindrical hole’s radius will cause press fit situations along the disassembly path. Hence all molded parts are made with a small draft angle, to assist the release of the part from the mold.

A draft will convert a cylindrical shape into a conical shape. When dealing with cylinders, drafts are also called tapers. For all other shapes, we call such shape modifiers as drafts.

Specifying a draft:
Specification of a draft feature typically requires specifying (a) which face or faces need to be drafted, (b) a neutral face (or neutral plane) – that is, a face which intersects the faces being drafted, such that the cross-section of the drafting faces on the neutral face remains unchanged before and after the drafting, and (c) the draft angle
These terms need to be properly defined based on the different geometric cases; some examples are shown below.



This is a fairly simple example of a taper (draft). Notice that the cross section area of the neutral plane remains constant in each of the three cylinders; therefore, the size of the cone changes. The draft angle here is the cone half-angle.



There are four draft features in the above example. The first three are simple – two opposite faces on each rib are drafted, each time using a different neutral plane (in order, the top face, the bottom face, and a face in the middle of the rib). The cross-section area of the neutral plane remains constant. The draft angle is measured as the angle by which the draft face is rotated, measured in the plane formed by the normal vectors of the draft face and the neutral face.

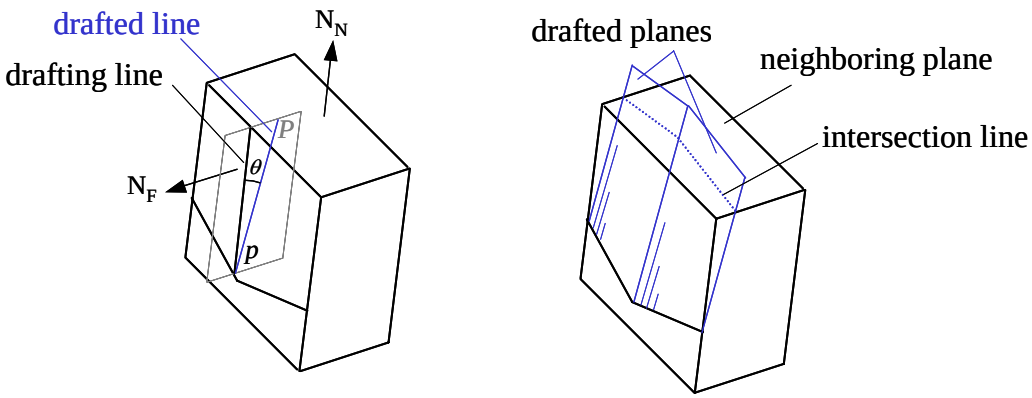
Another way to specify a draft is by specifying (a) the *drafting edge* (which can be a single edge, a sequence of edges, or a loop) that must remain fixed, (b) the faces adjacent to the selected edge(s) that must be drafted, and (c) the draft angle. This is shown in the example of the fourth rib in the above figure, where the V-shaped green lines are the drafting edge, and the portion of the face above the V must be drafted by the draft angle= 15 degrees (measured with reference to the neutral top face shown in green).

How a solid modeler handles a drafting request:

Drafting and tapering operations can involve of several different cases that one must study. We shall ignore these and just try to understand one fundamental type of drafting operation.
(i) Intersect the neutral plane with the draft faces, to get a series of edges (or a loop) that must stay fixed;
(ii) Each point, *p* on the fixed loop belongs to a Draft face, F.
(iii) For each point *p*, form the drafted edge:

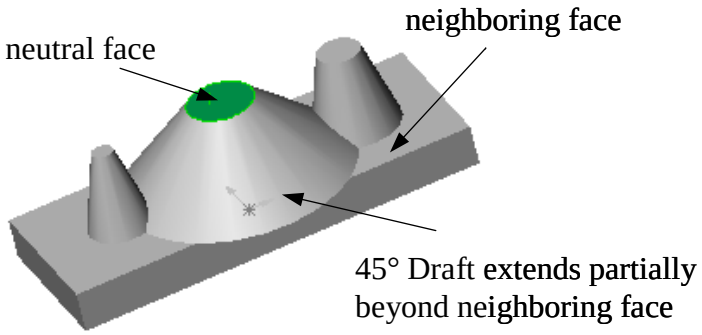
(a) *P* = plane through *p*, with normal = N_F X N_N
(N_F is normal to F, N_N is normal to neutral face)
(b) Intersect P with F to get drafting edge, e_p.
(c) Rotate e_p by the drafting angle in plane *P*, to get the drafted edge.

The drafted surface is formed by the extension of the surfaces formed by all the drafted edges. The boundary of the drafted surface is determined by finding (a) the intersection of the drafted surface with the part faces neighboring the draft face, and (b) the intersection of the drafted surface with its neighboring drafted surface(s). The figure below shows the process.



Once the draft surface boundaries are determined, the draft feature shape is completely determined, so a Boolean operation will provide the resulting shape. Note that the Boolean operation may be Union (if the draft surface intersects the neighboring face along an edge outside the neighboring face), or a difference (intersection line is inside the neighboring face, as in the case above).

When does a draft attempt fail:
(a) If the draft feature results in generation of non 2-manifold geometry in some region.
(b) In several steps, intersection of faces is required; if such intersection fails, or if the intersection edge is partly inside, partly outside the face, then there is a likelihood of failure to generate the geometry.
In the second case, some modelers may still be able to generate the feature successfully, as seen in the example below.
(c) Another case when drafting often fails is similar to the example below, but where the neighboring face is not planar. In this case, the drafted face must be intersected with an extension of the neighboring face; since the face is non-planar, it may not be clear how to extend it. Another possibility is that the blending face cannot intersect at all with the neighboring face (for example, if the neighboring face is cylindrical).



7. Surface offsets

Typically, each CAD kernel will incorporate a special library of functions related to surface manipulation. These functions handle complex surfaces – those that are defined by some parametric polynomial forms such as B-Splines or NURBS – and perform tasks like generating an offset surface to one or more surfaces, surface-surface-intersections (SSI) etc. Computing offsets of surfaces is an important function, even for surfaces based on conic sections. It is used often by CAD designers, and is also used internally by other CAD functions such as shelling.

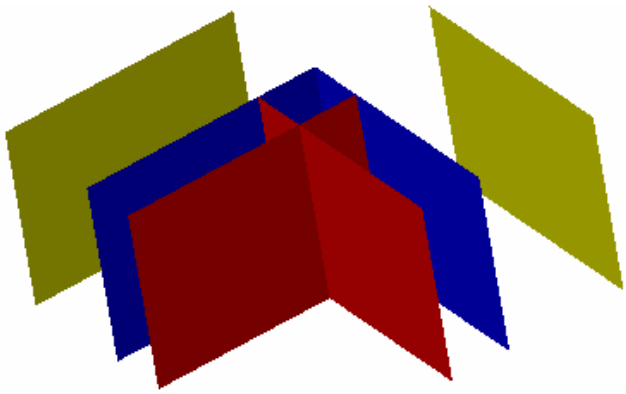
Two things must be specified to generate an offset surface: the base surface, and the offset distance. Let the unit surface normal vector at any point $\boldsymbol{p}$ on the base surface be $\boldsymbol{N}_p$. Then the offset point of $\boldsymbol{p}$ is given by the vector $\boldsymbol{p} + r\,\boldsymbol{N}_p$, where r is the offset distance. The set of offset points for all the points of the base surface form a surface, called the offset surface.
Let us look a four representative cases:
(a) Single planar face: the offset of the face is obtained by simply performing a translation on each vertex (that is, each vertex of each co-edge bounding the face), by the vector $r\boldsymbol{N}$, where r is the offset distance, and $\boldsymbol{N}$ is the face unit normal vector.

(b) Single cylindrical face: The offset is also a cylindrical face. The boundary of this face is computed as follows: let $\boldsymbol{p}$ be a point on the boundary of the cylinder, whose axis is along the unit vector $\boldsymbol{U}$, through some point on the axis, $\boldsymbol{a}$. The orthogonal projection of $\boldsymbol{p}$ on the axis is a point, $\boldsymbol{q} = (\boldsymbol{a} + x\boldsymbol{U})$. Then $(\boldsymbol{a} + x\boldsymbol{U} - \boldsymbol{p}).\boldsymbol{U} = 0$, which gives $x = (\boldsymbol{p} - \boldsymbol{a}).\boldsymbol{U}$. The corresponding point on the boundary of the offset is therefore $\boldsymbol{q} + r(\boldsymbol{p} - \boldsymbol{q})/|\boldsymbol{p} - \boldsymbol{q}|$.

Exercise:
Find the offset of a cylindrical surface of radius R, *inwards*, by an amount r? What happens when $r = R$? What if $r > R$?

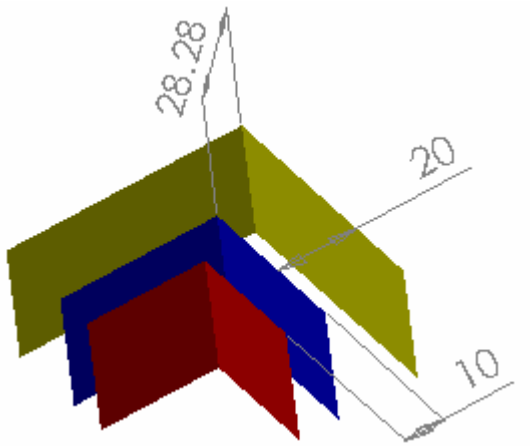
(c) Single B-spline surface: This is a problem case, since most CAD systems are able to handle general surfaces using a uniform representation scheme, e.g. B-Splines or NURBS. Unfortunately, the offset surface of a B-Spline (NURBS) is not a B-Spline (NURBS), and such a surface cannot be handled by the CAD system. Therefore most CAD systems will settle for an approximate solution:
(i) uniformly distribute a large number of points on the surface
(ii) for each point, $\boldsymbol{p}$, find the unit normal vector, $\boldsymbol{N}$, and the offset point, $\boldsymbol{p} + r\boldsymbol{N}$.
(iii) Interpolate a surface passing through all the offset points (this requires solving a large system of linear equations).
This approximation can be improved by increasing the number of points. Another problem of using this method is that while the original surface may be well-behaved, the offset surface may intersect with itself. In that happens, the offset operation may fail.

(d) Multiple connected faces: The first step in this case is to individually offset each face, using the methods above. However surface may not be smooth across a shared edge of the connected faces, so special care needs to be taken. Let us see what happens to offsets of two connected faces on the concave side and on the convex side.



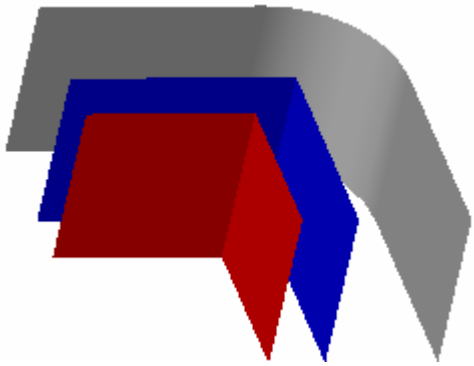
The figure above shows two connected planar surfaces (blue color). The red faces are the offsets of the faces in the concave direction. The yellow faces are the offsets in the convex

direction. In one case, the offset faces intersect, in the other, they are disconnected. Ideally, we would like the offset of a set of *connected* faces to be a set of *connected* faces. How do CAD systems handle the conditions above ?
(i) If the offset faces intersect, then their intersection is first calculated, and some part of the offset surface is deleted, using an operation called surface trimming.
(ii) If the offsets are disconnected, then the individual offset faces are extended using the underlying geometry of the face (in this case, the underlying geometry is a plane for both surfaces). The intersection of the extended faces is found, and the connected surface is found accordingly. This is shown in the figure below.



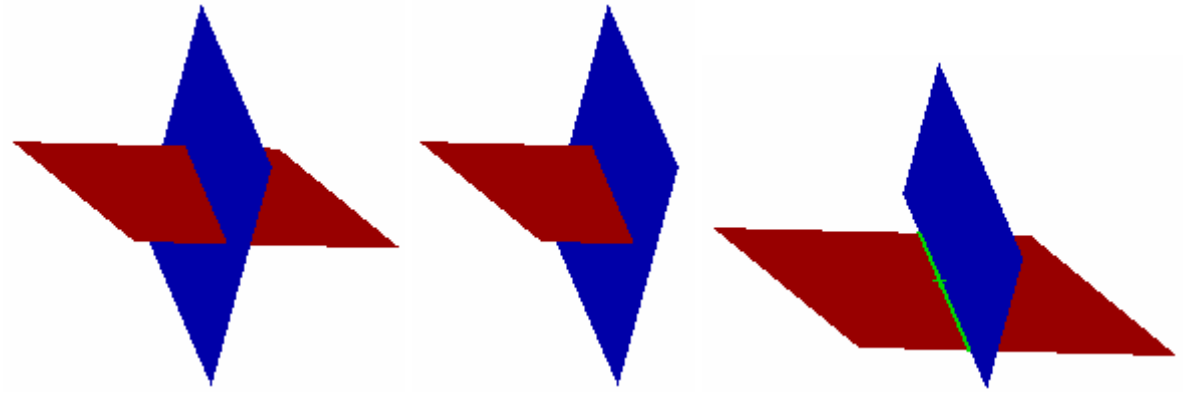
If the faces have complex shapes (e.g. B-Spline surfaces), then it is quite difficult to compute the intersection of the offset surfaces. Also, the offset faces may intersect several times, in several places, resulting in complicated cases. Very often, the operation will fail in such cases. Another problem is the extension of offset faces that are not connected. Most CAD systems will attempt to merge the disconnected edges using a surface that has some level of continuity at the edges. Even more complex is the case where the offset surfaces intersect in some regions, self-intersect in others, and also are disconnected at some parts.

A final note about the face extension as seen in the above figure. Due to the convention of extending surfaces using underlying geometry, the convex side offset is not a true offset – the shortest distance between two points along the shared edge is 28.28 units, rather than 20. In this sense, the true offset may be a surface that looks like the one in the following figure; but that is not the way that most CAD systems work.

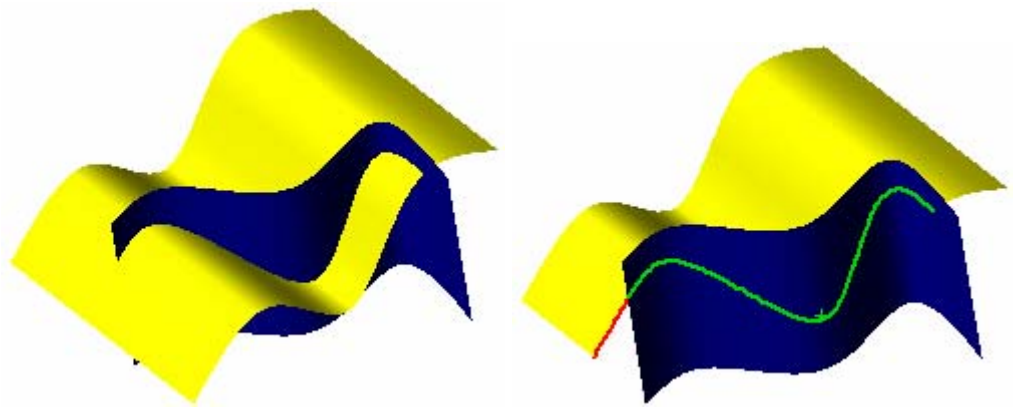


8. Trimming

Trimming is the operation of cutting away a part of an entity. There must be some method to partition the entity into two parts -- a part to be kept, and the remaining part, which is to be cut. This partition is usually found by the intersection of the entity with another entity. Hence, the most important geometric functions that are used in trimming are (i) intersection, and (ii) extrapolation. Trimming is quite useful in 2D sketching, where all entities are curves and lines, and their intersection is relatively easy to find. The difficult type of trimming is that between two faces, since several problem can arise. Firstly, it is not easy to compute the intersection of faces if they are complex. Secondly, the faces may intersect only partially, therefore not forming a proper partitioning. In that case, the intersection curve must be extended along the surface to be trimmed until the face is partitioned. This is done by extrapolation, which is fairly complex when the underlying face is complex. Some simple examples of surface trimming are discussed through examples below.
Example 1. Figure (a) shows a red face partitioned by the blue face. Figure (b) shows the remaining part of the red face if we use the blue face to trim the red face. Figure (c) shows the reverse case – if we use the red face to trim the blue face. Here, the intersection edge (in green color) must first be extrapolated along the blue face to complete the partitioning. Then the trimming operation can be performed. Of course, in each trimming operation, one must also specify which partition must be kept, and which is to be discarded.

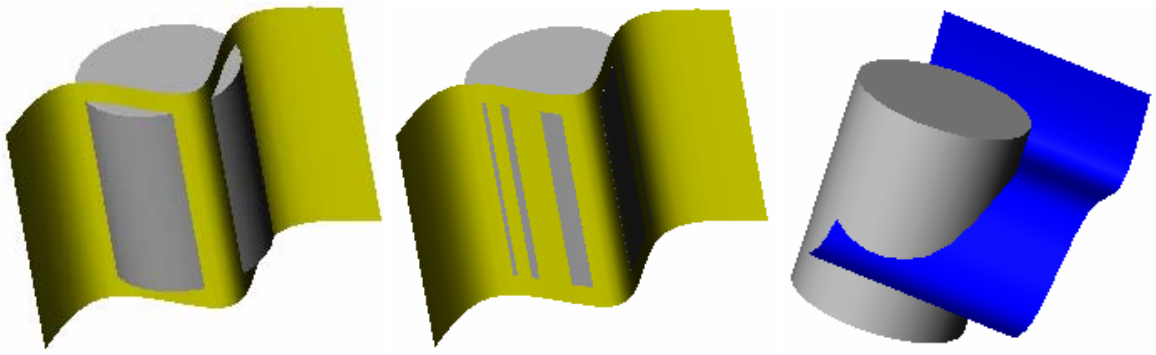


Example 2. Here, we see two B-Spline surfaces (yellow and blue). If we use the blue surface to trim the yellow one, then the intersection line (green) must be extrapolated on the yellow surface to partition it completely. Here, the extrapolated curve is shown in red.



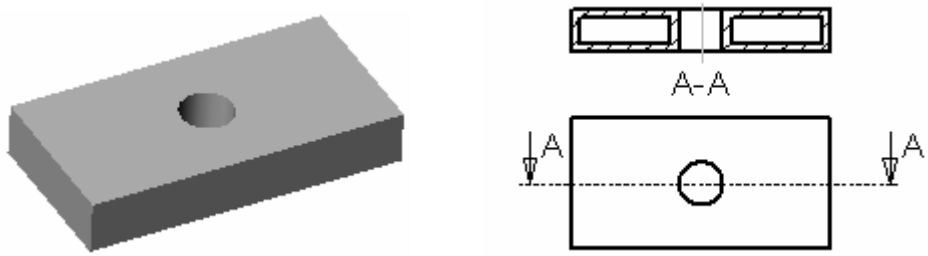
It is easy to see that once an operation to trim one surface with another is well defined, it can be extended to allow trimming of a solid by the use of a surface. In other words, to use a

surface to cut away a portion of a solid. Such trim operations are often useful in CAD; as usual, they will fail if the resulting solid is non 2-manifold. Another possible reason is that the surface may not be partitioning the solid into two parts. In the examples below, the cylinder can be trimmed using the yellow surface, but cannot be trimmed using the blue surface unless the blue surface is extrapolated.

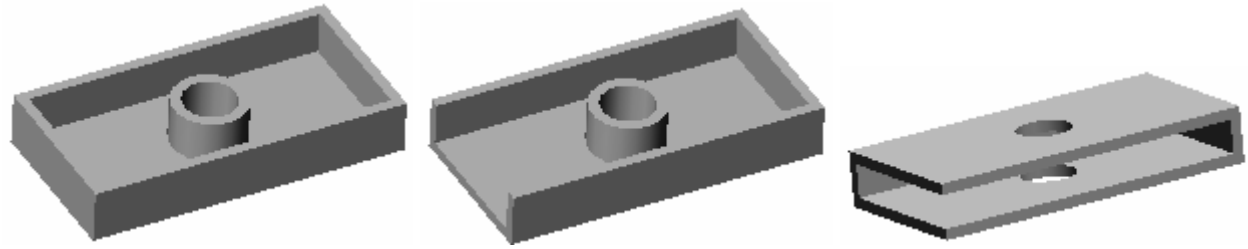


9. Shelling

Shelling operator converts a solid into a shell. For example, you could get the skin of an apple if you shell it to a small thickness. The operation involves first computing the inwards offset surfaces for all the faces that need to be shelled; next the intersection of these faces is computed to determine the edges (and therefore the vertices) of the inner surface of the shell. These faces, edges and vertices are used to update the solid model of the part, converting it from a 'filled' solid to a 'hollow' solid.
The figure below shows a simple example of a part that was shelled; it is clear from the section view of the part how the shelling makes the inside hollow.



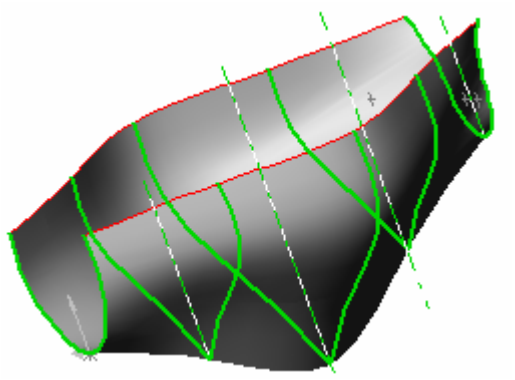
In most practical cases of shelling, we may specify if a face on the solid may not be offset during the shelling operation. Most of the inner region of such faces will thin be cut away, leaving just a ring around the border. This is because the offset of the adjacent faces will cause the interior of such faces to be trimmed when the shell is performed. The figure below shows different examples of this, when (a) the top face is not used for shelling (i.e. is removed); (b) both the top face and left side face are not used; (c) the left side, front and the cylindrical faces are not used for shelling.



(a) (b) (c)

10. Lofting

Lofting is probably the most complex of all CAD operations. However, it is extremely useful to designers since it provides the mechanism to construct complex 3D shapes by only specifying two or more cross sections of the shape.
Typically, if a complex 2D contour is extruded or swept, we get a 3D complex shape with a constant cross section. However, in many designs, the shape may change across different cross-sections. In such cases, a lofting operation is required.
The figure below shows a surface that is constructed by lofting between the five (cross-section) curves shown in green color.



Obviously, there are infinite number of surfaces that can pass through the cross sections shown. Which one will the lofting operation select?
Firstly, the selection is restricted to surfaces that can be described by means of the surface type that the modeler uses (e.g. B-Spline based modelers will most like use a piecewise polynomial representation of degree 2 or 3 in both u- and v- parameters for surface $\boldsymbol{S}(u, v)$).
Secondly, the designer can further restrict the lofting operation by specifying additional curves, called guide curves. A guide curve is a smooth curve going through a point on each of the cross-section curves. The guide curves used in the above example are the two red edges along the top of the surface.

Notes for ielm317, HKUST, Ajay Joneja