4

Solid Object Representation *(Phenotypes)*

4.1 Introduction

The representation of phenotypes is perhaps the most fundamental of the four elements of the design system. The definition of how the system can represent designs, defines all allowable solutions that can be generated by the system. In other words, the phenotype representation enumerates the search space, defining all permissible designs as points in that space. Moreover, since all phenotypes must have a corresponding genotype to allow manipulation by a GA, the phenotype representation plays a significant role in determining the size and complexity of the genotype.

For this work, phenotypes are solid object designs. Since the evolutionary design system is intended to have the capability to evolve a wide range of such designs, the phenotype representation must be capable of defining a wide range of solid object designs. Significantly, evolving designs from scratch requires a very different approach to the representation of designs when compared to more traditional systems, e.g. systems that optimise existing designs. Typical design optimisation systems only optimise selected parameters (e.g. for a jet-turbine blade, such parameters define the angle and length of the blade). To allow a GA to create a new design, the GA must be able to modify more than a small selected part of that design - it must be able to modify every part of the design.

This solid object representation must be designed to allow efficient evolution by a GA. Unlike the requirements of a representation used in computer aided design (CAD) packages, where shapes must be easily modified using as few inputs from the user as possible (Foley et. al., 1990), a representation intended purely for a GA should be designed to allow any one design to be altered to any other design in a series of continuous, small steps.

Consequently, the investigation and evaluation of existing solid object representations and the creation of a new representation suitable for manipulation by a GA, formed the first stage of the development of the design system.

4.2 Choosing a Representation

4.2.1 Requirements of the Representation

Using evolutionary search to tackle design problems imposes certain restrictions and requirements on phenotype representations. A representation that describes a design in terms of many vertices per measurement unit would require huge numbers of parameters to define even the simplest of shapes. The more parameters in the phenotype, the more genes there are in the genotype, making the search problem that much larger and more difficult. (The effective search-space, a subset of the full design-space, is defined by the number of parameters currently being examined by the search algorithm. Five parameters define a five-dimensional search-space or hyperspace.) So the phenotype representation must be capable of adequately defining a shape using a low number of parameters.

The representation must allow the easy modification of the shape it defines (i.e. 'easy' for the computer to modify for the purposes of evolution, not necessarily easy for humans). More precisely, this ease of modification ideally should allow any design to be changed in any way, to any degree. Evolution demands small, gradual changes, so an enumeration that places very dissimilar designs 'side-by-side' in the space (forming discontinuities in the search space) would make that space more difficult to traverse. If a represented design could not be gradually refined, then only a large and highly improbable mutation could ever allow the GA to locate a good design. By using a representation that always places similar designs next to each other in the search space, the GA should always be able to find an evolutionary path from poor designs to better designs.

The representation need not be unique, i.e. any given design can be defined in more than one way, but must be complete or unambiguous - the representation must only define one design at a time. As mentioned previously, the representation must be able to define a wide range of three-dimensional solid shapes, so should have an infinite domain (although in reality, the memory of the computer will make the domain finite). Ideally, it should be accurate, i.e. objects represented without approximation.

4.2.2 Existing Representations

There are two main types of suitable existing representations: those defining the shape of surfaces only (with the assumption that the space enclosed by the surface is solid), and those defining the shape of solids more directly (Foley et. al., 1990). Surface representations (or boundary representations), typically use combinations of equations and control points to specify shapes, and all suffer from similar drawbacks. Many define curves with great accuracy, and most define non-solids with ease, but they often require the use of patches (many separately defined segments of surfaces joined together) to define solids fully. The number of parameters needed to specify even the simplest of shapes is very high and changing the level of detail of designs specified by these representations is often over-complicated and computationally expensive. Examples of surface representations include: Polygon Mesh, Quadric, Hermite, Fourier (Foley et. al., 1990), Parametric (Joy, 1991), Bézier, and B-Spline (Anwei et al., 1989).

Solid representations are perhaps a better choice for the purposes of the evolutionary design system, for they can define not only solids with good accuracy and few parameters, but also surfaces and 2D shapes. There are three commonly used types of solid representation: sweeping, constructive solid geometry, and spatial partitioning.

The first of these define objects by 'sweeping' a two-dimensional object along a trajectory (extruding) or rotating it around an axis. However, sweep representations are not suitable for complex designs - combining swept objects is normally only performed after converting to an alternative representation (Foley et. al., 1990).

The second commonly used type of representation, known as constructive solid geometry (CSG), is used extensively in CAD packages (Chuan Jun Su et al., 1991). CSG combines different primitive shapes to form more complex shapes. Although it is an efficient representation capable of handling 2D as well as 3D designs, because of the use of dissimilar primitives, the design space is not enumerated in the most convenient way. For example, consider the solid defined by the intersection of a sphere and a cylinder, fig. 4.1 (left). If mutation was to change the cylinder to a plane, the resulting solid is very different, fig. 4.1 (right). In other words, CSG places dissimilar shapes close to each other in the search space, making the traversal of such a space difficult. If such a representation was used, it seems likely that after the first few generations the GA would not explore the dimensions corresponding to changing the type of primitives. Having settled early on certain types of primitives, and optimised the shape of these primitives, the GA would be unable to change one primitive into another without reducing the fitness of a design.

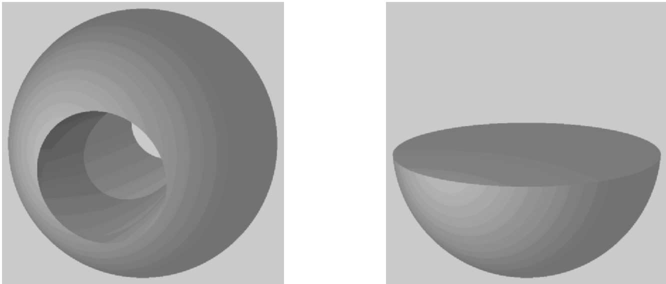


Fig. 4.1 Dissimilar shapes next to each other in the search space defined by CSG.

However, the third of the commonly used solid representations, spatial partitioning, seems ideal. All types of spatial partitioning representations involve decomposing a solid into a collection of smaller, adjoining, non-intersecting solids that are more primitive than the original solid. There are a number of variations including: cell decomposition, spatial-occupancy enumeration, octrees, and binary space-partitioning trees (Foley et. al., 1990).

The advantages of spatial partitioning when compared to other representations are clear. It is easy to modify by small increments and can define any 3D shape, albeit sometimes by approximation. It is not a unique representation, but that is not seen as any disadvantage - it may even help the searching process (i.e. if the same solid can be represented by different primitive shapes, it will appear in more than one place in the search space, making it potentially easier to find). It is complete and is a very modular representation highly suited to coding in a chromosome. All the other representations considered have either limited domains, are difficult to modify easily or require too many parameters.

Although the design-space is enumerated in a very convenient manner (normally the more different two designs are from one another, the 'further' they are from each other in the space), spatial partitioning is not the most efficient in terms of the number of parameters needed. If the parameters could be reduced to keep the search-problem as small as possible, this representation would allow the steady refinement of designs.

4.2.3 Reducing Parameters

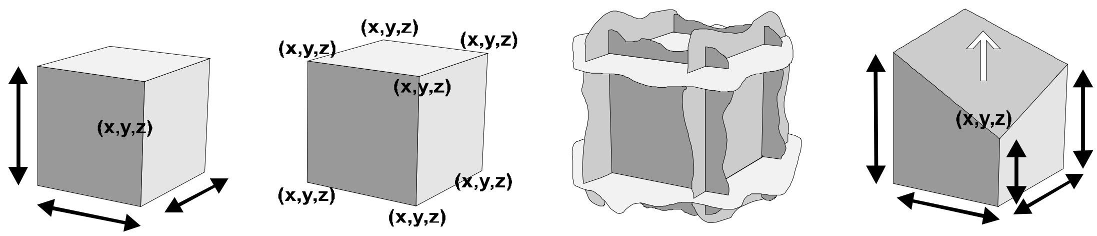
There are two methods that can be used to reduce the number of parameters in a representation. The first is to create a new version of the representation, designed to have a minimum number of explicit definition parameters. The second method is to use some form of 'artificial growth' process with an existing representation. As was described in Chapter Two, the latter method is often termed 'artificial embryology', i.e. the use of a 'shape grammar' to specify how shapes should be artificially 'grown' using a given representation, instead of explicitly defining the shapes. Todd used such a technique with a CSG representation to allow structures such as horns and tentacles to be defined recursively (Todd and Latham, 1992). Sims also used a recursive method with a representation similar to spatial partitioning to define his 'virtual creatures' (Sims 1994a, 1994b). Gero, Rosenman and de Garis used shape grammars with spatial partitioning in attempts to define two-dimensional shapes (Gero and Louis 1995, Rosenman 1996, de Garis 1992).

Unfortunately, however attractive it may seem to improve the analogy with nature by incorporating the concept of artificial embryology, this technique does create some difficulties. Recursively defined shapes are very efficient at representing long repeating structures such as the 'ribs' in Todd and Latham's evolved art and the 'artificial snakes' of Sims evolved creatures, but such methods are not always suitable for more general shapes. Moreover, as described by Rosenman, a small change in the genotype can lead to large changes in the phenotype (Rosenman, 1996). In other words, if just a single parameter value is altered by the GA, the resulting shape can be radically different. This seems to be a common flaw with many representations using 'artificial embryology' - insufficient attention was paid to the effective enumeration of the search space, resulting in many discontinuities, with very dissimilar shapes placed side-by-side in the space. Rosenman argues that this is excusable, since the GA is capable of successfully finding solutions for some discontinuous functions (Rosenman, 1996). However, there seems little reason to make the enormous problem of searching for designs more difficult if it can be avoided. As de Garis found when he was barely able to evolve even a simple two-dimensional 'L' shape, these difficulties can be significant (de Garis, 1992). Hence, although artificial embryology could be used to significantly reduce the parameter count of spatial partitioning representations, this work uses the more conventional alternative - the creation of a new, low parameter spatial-partitioning representation.

4.2.4 Low-Parameter Spatial Partitioning

Spatial partitioning defines solids by partitioning the solids into a number of smaller, non-intersecting shapes. This means that the number of parameters needed can be reduced by minimising the number of such primitive shapes needed to define designs, and by reducing the number of parameters needed to define each shape (Bentley & Wakefield, 1996b).

It seems appropriate to permit variable sizes of partitions so that a large space can be taken up by a single large primitive shape rather than many smaller primitives. Ideally, the number of different kinds of primitives should be kept as low as possible to avoid discontinuities in the search-space.



A B C D

Fig. 4.2 From left to right: stretched cube, six-sided polyhedron using vertices, six-sided polyhedron using planes, stretched cube with moveable side.

Initially, four new types of spatial partitioning were considered, see fig. 4.2 (Bentley & Wakefield, 1996b). The 'Stretched Cubes' representation was the first and most simple of these new representations. It uses cubes with alterable lengths, widths and depths (see fig. 4.2 [A]), the solid being composed of a number of such stretched cubes, adjacent to each other and non-intersecting. However, despite only six parameters being needed to define each primitive shape, this representation is unable to approximate curved surfaces without using unacceptably large numbers of primitives.

Alternatively, the 'Polyhedra using Vertices' representation uses a number of six-sided polyhedra to define shapes. Each polyhedron is defined by its eight corners (see fig. 4.2 [B]), allowing curved surfaces to be closely approximated. Unfortunately, this representation requires twenty-four parameters per primitive shape (eight *x, y, z* points), which was considered unacceptably high.

A third representation, 'Polyhedra using Planes', was created in order to reduce this excessive parameter-count. This also defines shapes using six-sided polyhedra, but uses six intersecting planes to define each polyhedron, instead of eight corners, see fig. 4.2 [C]. The representation is just as flexible as 'Polyhedra using Vertices', and six fewer parameters are required per polyhedron. Nevertheless, eighteen parameters for every primitive shape is still excessive, and this representation suffers from problems of ambiguity (six intersecting planes can define more than one solid at once).

In an attempt to combine the best features from the low-parameter 'Stretched Cubes' representation and the flexible six-sided polyhedra representations, a fourth representation was created. The 'Stretched Cubes with Moveable Sides' representation allows each primitive to have a single side of variable orientation, the other five sides being defined as with the 'stretched cube' representation. By allowing the moveable side to point in any of six directions (left, right, up, down, forwards, or backwards), it is possible to use it to approximate a surface of any orientation. The primitive needs just nine defining parameters (*x, y, z, width, depth, height1, height2, height3, orientation*), the moveable side being specified by the three heights (three parameters defining a plane) and oriented in the direction specified by *orientation*, see fig. 4.2 [D]. However, there are two major disadvantages with this representation. Firstly, the problem of tessellation - a primitive totally surrounded by other primitives would leave gaps, unless the moveable side was fixed at right angles to the surrounding four sides. The second disadvantage is a little more serious, however. When representing objects with curved surfaces such as spheres, triangular gaps are left where no primitive can fit. Since this primitive must always have a rectangular base, the representation fails, see fig. 4.3.

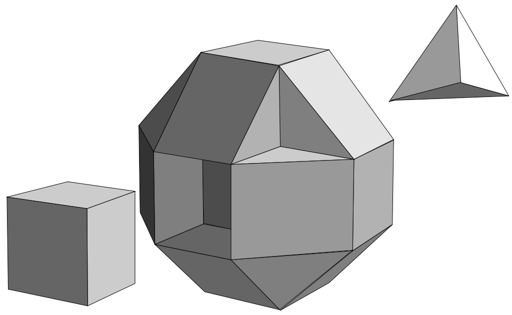


Fig 4.3 Four-sided polyhedron with triangular sides cannot be formed by primitive - representation fails.

4.2.5 Clipped Stretched Cubes

By examining the type of primitive required to represent varying curved surfaces (e.g. see top right of fig. 4.3), it becomes clear that the ideal primitive shape must be capable of having anything from four to seven sides of any orientation, must be easily modified, and must have very few parameters. The solution seems to be to use the 'stretched cube' representation where possible, and whenever a non-parallel surface needs to be approximated, allow the primitive to be sliced by a plane of the appropriate orientation. In this way, triangular versions of the primitive can be produced by slicing off a corner, and indeed, polyhedra from four to seven sides can be defined. The parameter count per primitive remains at nine, with the six parameters of the 'stretched cube' (*x*, *y*, *z*, *width*, *height*, *depth*) plus three more to define the plane (*angle1*, *angle2*, *distance of plane from centre*). Since the plane can be rotated to intersect any part of its corresponding primitive, no orientation parameter is required, see fig. 4.4.

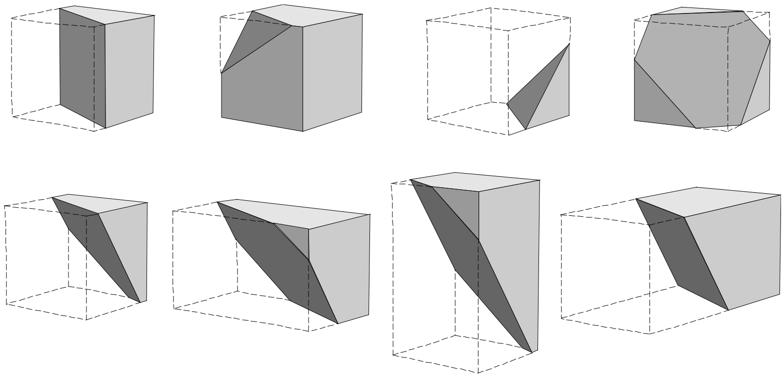


Fig. 4.4 Clipped stretched cubes.

This new representation can approximate closely any curved surface (see fig. 4.5), and with few parameters required per primitive, it is the most compact and flexible of all of the representations examined so far. Because of this, the 'Clipped Stretched Cubes' spatial partitioning representation was chosen for the generic evolutionary design system.

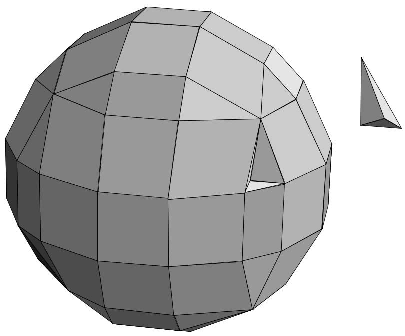


Fig. 4.5 An approximation of a sphere using the new representation.

In more detail, a 'clipped stretched cube' is a six-sided polyhedron with all sides at right-angles to each other, intersected by a plane defined relative to the centre of the polyhedron, fig. 4.6 (Bentley & Wakefield, 1996b). A solid is defined by having a number of stretched cubes in the centre surrounded by the clipped stretched cubes, which are only used for defining the surface of the solid. The representation is not unique in that the centre of a solid can be filled by many small primitives or a few large ones. Ideally, the solid should contain as few primitives as possible to minimise the total number of parameters. Therefore, to represent an existing design, every part of the design should be partitioned using the largest possible primitive, using stretched cubes and clipped stretched cubes of ever-decreasing size as less of the design remains unpartitioned. A stretched cube can be converted to a clipped stretched cube by the addition of three parameters that define the clipping (or intersecting) plane for that primitive. Even though primitives may be of varying dimensions, the representation is still a spatial-partitioning one, every primitive being one partition of the 3D space.

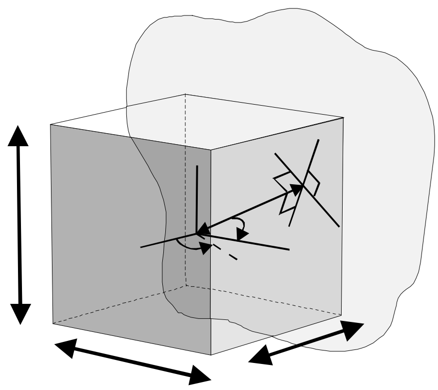


Fig. 4.6 Clipped Stretched Cube:

(*x*, *y*, *z*, *width*, *height*, *depth*, *planedist*, *angle1*, *angle2*).

As the reference point (*x*, *y*, *z*) defines the centre of the primitive, the eight vertices of the stretched cube are simply: ($x \pm width/2$, $y \pm height/2$, $z \pm depth/2$). The clipping plane is defined by a normal vector given by the two angles α and β, and distance d from the centre point: $\begin{pmatrix} \alpha \\ \beta \\ d \end{pmatrix}$

To obtain the familiar equation of a plane: $Ax + By + Cz + D = 0$ (Foley et. al., 1990)

the plane coefficients can be calculated as:

$A = d \cos\beta \sin\alpha$ $\quad B = d \sin\beta$ $\quad C = d \cos\beta \cos\alpha$ $\quad D = -(A^2+B^2+C^2)$

When calculating the shape of the primitive resulting from the intersection of the plane and the stretched cube, it is perhaps easiest to calculate the new vertices obtained (if any) when the plane intersects each of the twelve edges between the eight existing vertices of the stretched cube.

So, given an edge defined by two end vertices: V_1 (x_1, y_1, z_1) and V_2 (x_2, y_2, z_2) and a plane defined by the equation: $Ax + By + Cz + D = 0$, it can be calculated that the point: P_i (x, y, z) will be the point of intersection of the plane and edge, where:

$$x = \frac{Bx_1(y_2 - y_1) + Cx_1(z_2 - z_1) - (By_1 + Cz_1 + D)(x_2 - x_1)}{A(x_2 - x_1) + B(y_2 - y_1) + C(z_2 - z_1)}$$

$$y = \frac{Ay_1(x_2 - x_1) + Cy_1(z_2 - z_1) - (Ax_1 + Cz_1 + D)(y_2 - y_1)}{A(x_2 - x_1) + B(y_2 - y_1) + C(z_2 - z_1)}$$

$$z = \frac{Az_1(x_2 - x_1) + Bz_1(y_2 - y_1) - (Ax_1 + By_1 + D)(z_2 - z_1)}{A(x_2 - x_1) + B(y_2 - y_1) + C(z_2 - z_1)}$$

Intersection occurs between vertices V_1 and V_2 iff:

$x_2 \geq x \geq x_1$

$y_2 \geq y \geq y_1$

$z_2 \geq z \geq z_1$

The equations are invalid when $d = 0.0$ (with the plane passing through the origin, α and β are undefined, hence the plane itself is undefined). In practice, it is simple to add a negligible value to d to overcome the problem without any significant loss of accuracy.

Once any additional intersection vertices have been calculated, the vertices clipped by the plane can be removed by examining the distance of each vertex from the plane. If the distance is positive, (or negative if *planedist* < 0) the vertex is on the outside of the plane and should be discarded.

So, given a vertex: V(r, s, t) and plane: $Ax + By + Cz + D = 0$, the distance from the vertex to the plane is given by:

$$Dist = \frac{Ar + Bs + Ct + D}{\sqrt{A^2 + B^2 + C^2}}$$ (Foley et. al., 1990.)

However, since only the sign of the distance is required, it is sufficient to use:

$Dist = Ar + Bs + Ct + D$

The remaining vertices are the corners of the primitive, and define the size and orientation of its sides. The algorithm for extracting the positions of the sides and vertices of a primitive from the nine definition parameters within a phenotype is given in Appendix A.

4.3 Using the Representation

4.3.1 Positioning Primitives

Within the evolutionary design system, the phenotype of an individual solution specifies the geometry of that design using the 'stretched clipped cubes' spatial partitioning representation. As described above, this representation defines the shape of solids by using a number of non-overlapping primitive shapes in combination. These primitive shapes all have their positions determined relative to a global origin (which typically is at, or near to, the centre of each solid).

This method of independently specifying the position of each primitive was chosen for a reason. Whilst it would be quite possible to specify the position of every primitive relative to neighbouring primitives and perhaps reduce the number of parameters required, this would reduce the freedom of the system to manipulate the shape of the designs. During evolution, the system uses mutation operators to add or remove new primitives in designs. If the position of primitive B was based upon a translation from primitive A, the system would be unable to remove primitive A because then primitive B would have no way to determine its position. Moreover, moving any primitive that others base their position on, would also move all the others, i.e. the independent movement of single primitives would be impossible without corresponding adjustments. By specifying the position of every primitive independently of each other, these problems are resolved.

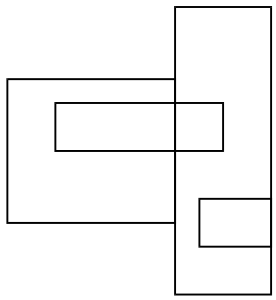


Fig. 4.7 An illegal design with redundant overlapping primitives.

However, one problem is caused by such explicit independent positioning of primitives: when the positions are permitted to vary randomly during evolution, it is possible for primitives to overlap each other. A spatial partitioning representation cannot have an overlapping partition of space: it is meaningless and causes unwanted redundancy in the representation. If primitives are completely inside other primitives, they are redundant, playing no part in the definition of the solid, see fig. 4.7. Thus phenotypes with overlapping primitives, which constitute illegal designs, must either be prevented or corrected.

4.3.2 Correcting Illegal Designs

To detect whether a design is illegal, every primitive must be compared with every other primitive in the design. After some investigation (Bentley & Wakefield, 1995a/1996a), a quick and efficient conditional check was created, fig. 4.8.

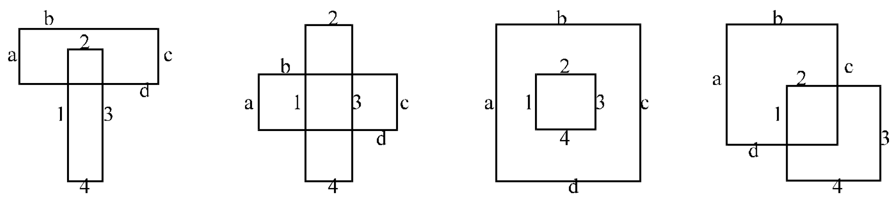


Fig. 4.8 If side *1* or *3* lies between *a* and *c*, or if side *a* or *c* lies between *1* and *3*, and if side *2* or *4* lies between *b* and *d* or if side *b* or *d* lies between *2* and *4*, then the two primitives are overlapping.

Once an illegal design has been detected, there are three options available: correct its genotype, correct its phenotype, or prevent it ever from existing in the population.

To deal with the last option, first: if every time the GA creates an illegal design, that design is discarded, the GA takes substantially longer to evolve designs, since illegal designs are often evolved. When simple tests are performed, it becomes clear that the more primitives there are in parent designs, the more likely it is for the offspring to have overlapping primitives. Indeed, for some unfortunate combinations of two parents, the offspring *always* have overlapping primitives, so the system simply gets trapped in an endless loop of creating and discarding illegal child designs.

Alternatively, if the illegal designs are allowed to exist in a population, but have their genotypes corrected (i.e. 'genetic engineering'), the population of candidate designs becomes very static, with little evolution occurring. This seems to be because a high proportion of all new designs generated by crossover and mutation are illegal designs - if these are corrected every time, evolution is effectively being undone so the population stagnates (Bentley & Wakefield, 1995a/1996a).

So the best solution seems to be to correct the phenotypes. This allows evolution to continue unconstrained, and does not reduce the speed noticeably or allow the population to stagnate, whilst still always ensuring that only legal phenotypes exist. Hence, the system allows encoded illegal designs to exist as genotypes, but corrects the designs when they are mapped to the phenotypes.

Two overlapping primitives in a design are corrected by examining the positions of the primitives relative to each other. If the difference in position between the two primitives is greatest in the x direction, the widths of the primitives are reduced, if the difference is greatest in the y direction, the heights are reduced, and for a greatest difference in the z direction, the depths are reduced, see fig 4.9. Using the positions to determine which sides to move ensures that the primitives are changed by the minimum amount possible. When reducing the dimensions, they are reduced until instead of overlapping, the appropriate sides are touching.

To prevent too much unnecessary squashing, all other sides of the primitives are kept in the same place by moving the centre positions of the two primitives slightly.

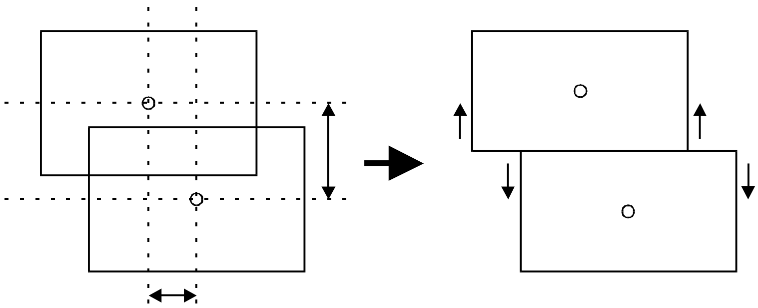


Fig. 4.9 Distance between overlapping primitives is greatest along y-axis: heights (and y positions) are changed.

Consequently, the genotype no longer directly specifies the phenotype (or design), but instead gives directions that should be carried out if permissible. This method corresponds well with nature. For example, suppose that there is a single gene within a person's genotype that specifies how tall he should be (in reality it is probably a collection of genes). It might 'say' his ideal height should be 6'5" - but in real life he might only be 6'. Why? Because whilst growing, he was restricted by gravity. If he had grown up in space without the restriction of gravity, he may well have become 6'5".

Equally, the designs are restricted by the rules of the representation. Even though in the genotype, a primitive shape may be given a large width, when the genotype is mapped to the phenotype, that primitive may be squashed to a smaller size and new position by its neighbours. It is apparent that in nature, evolution takes into account such restrictions - our genes compensate for gravity when specifying our heights, so although the human genes may be 'asking' for a height of 6'5" they 'know' that, because of gravity, this equates to 6'. Equally, as will be shown by the successfully evolved designs later in this thesis, the evolution of the GA takes into account the restriction of the design representation, and compensates.

It should be noted that the intersecting planes of 'stretched cubes' are ignored during the detection and correction processes. In other words, this representation forbids overlapping partitions of space, whether that partition has been intersected by a plane or not, fig. 4.10.

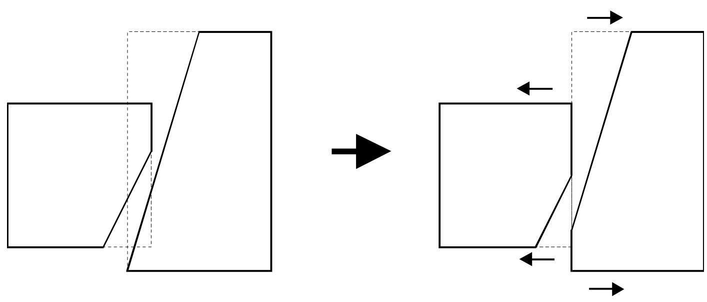


Fig. 4.10 Primitives are 'squashed' regardless of their intersecting planes.

Intersecting planes are ignored for two reasons. First, calculating exactly where two intersected primitives overlap and what degree of correction is required is substantially more computationally expensive than simply ignoring the intersecting planes. Second, the only occasion on which this can cause problems is when an intersecting plane faces another primitive, as shown in fig. 4.10. Typically, in this situation the correction process will detach a primitive from the design. As described previously, the intersecting planes are intended to be used to represent external surfaces of designs, so primitives with internally-facing planes should be discouraged anyway. Since the system is intended to be used to evolve a single design at a time, most designs with detached primitives are penalised by an evaluation module. Hence, by selecting this commonly-needed evaluation module, designs defined by such primitives can be indirectly discouraged, fig. 4.11. In other words, ignoring intersecting planes during the correction process is not only faster, but can also help to encourage the correct use of the representation.

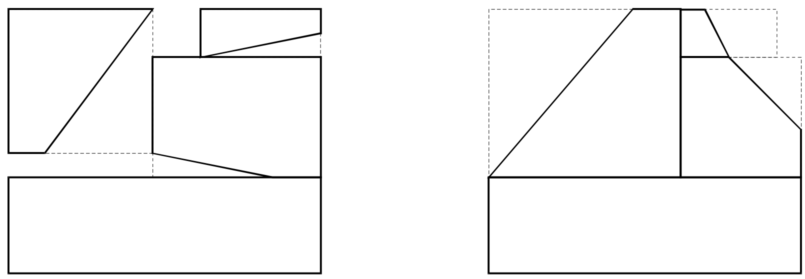


Fig. 4.11 Designs with internally facing planes (left) are discouraged in favour of designs with externally facing planes (right).

In conclusion, although the need for such a correction process may seem slightly cumbersome, the process itself uses negligible amounts of computation time, even for designs defined by large numbers of primitives. The process can be thought of as a simple type of artificial embryology, allowing genotypes to indirectly define legal phenotypes without any detrimental effect on evolution. When used in combination with the appropriate evaluation module, the process encourages the proper use of the representation. Moreover, as will be described in the following section, the correction process is also useful when symmetrical designs are being generated.

4.3.3 Symmetrical Designs

It is a well-known fact that symmetry has a powerful influence on the production of good designs in nature (Hawkes, 1995). Equally, it is clear that many human designs are symmetrical along at least one axis. Early work evolving designs with the system (tables that were plainly asymmetrical) soon confirmed the need for the system to have the capability to evolve symmetrical designs (Bentley & Wakefield, 1995b).

Symmetrical designs could be requested by the addition of a new module of evaluation software, i.e. the less symmetrical a design is, the less fit it is. However, the more primitives a design has, the more difficult the task becomes. The best that could be hoped for would be an approximation to symmetry - the GA would rarely evolve designs that fully met the strict requirements of exact symmetry. Additionally, the calculations to determine the degree of symmetry in a design would be complex and computationally expensive.

A more attractive method is to enforce symmetry by reflecting the design in one or more planes. This reflection is performed when mapping the genotype to the phenotype, meaning that the genotype need only specify the non-reflected portion of the design. This has the advantage that symmetrical designs are always produced, and that the GA only needs to manipulate half or a quarter (depending on the number of reflections) of the primitives in each design.

The system allows reflection to be performed in the x = 0, y = 0 and z = 0 planes. A design can intersect a plane and still be reflected in it: all primitives on one side are reflected to the other (a reflected primitive is identical to the original except for a changed position and re-oriented clipping plane). Since this can (and often does) produce designs that have primitives overlapping, the reflected design must be corrected once again.

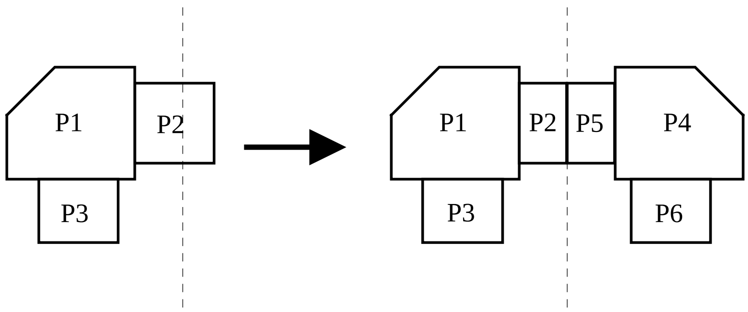


Fig. 4.12 Reflecting partial design P1 - P3 in the plane X = 0

to produce symmetrical design P1 - P6.

	P1	P2	P3
P4	1*	2	4
P5	2	3*	5
P6	4	5	6*

Table 4.1 Order to check and correct overlapping primitives

(after reflection in x = 0).

For example, consider a design consisting of primitive shapes P1 to P3. The genotype of the design will always only hold coded versions of these three, no matter how many reflections take place later. During the mapping of genotypes to phenotypes, the three primitives are checked against each other and corrected should any overlap. The design is then reflected in the plane x = 0, producing a symmetrical design consisting of primitives P1 to P6, see fig. 4.12. The two halves: P1 to P3 and P4 to P6 are then checked for overlaps and corrected if necessary. This checking and correction process must be performed in a symmetrical manner, to ensure both halves of the design are corrected identically. As described previously, to correct overlapping primitives, the primitives are squashed in the direction which ensures the least change occurs to them. However, there is a special case when correcting a primitive that overlaps its mirror image - it must always be squashed in the direction normal to the plane of reflection. Table 4.1 shows the order in which the checks must be performed, with each matching number representing a check to be performed simultaneously and each special case denoted by an asterisk.

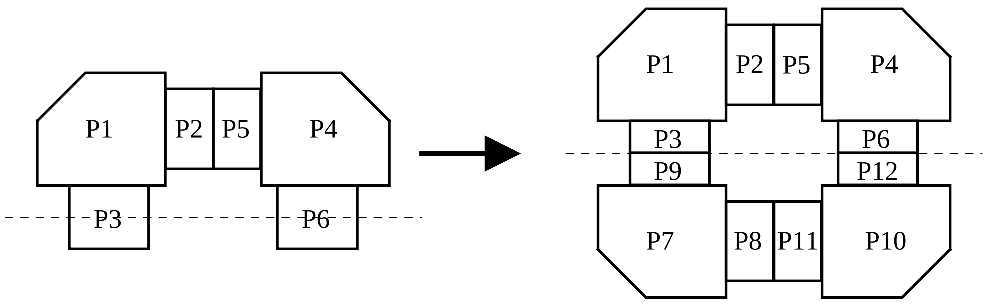


Fig. 4.13 Reflecting partial design P1 - P6 in the plane Z = 0

to produce symmetrical design P1 - P12.

	P1	P2	P3	P4	P5	P6
P7	1*	2	4	1	2	4
P8	2	3*	5	2	3	5
P9	4	5	6*	4	5	6
P10	1	2	4	1*	2	4
P11	2	3	5	2	3*	5
P12	4	5	6	4	5	6*

Table 4.2 Order to check and correct overlapping primitives

(after reflection in x = 0, then z = 0).

For a design symmetrical in two planes, the process is repeated. The design consisting of primitives P1 to P6 is reflected in z = 0, see fig. 4.13. The new design P1 to P12 must then be corrected again, with these checks and corrections (if necessary) occurring in a new symmetrical manner as shown in Table 4.2.

Despite the seemingly large number of checks that need to be performed for each design during the genotype to phenotype mapping, by checking at each stage, the number has been reduced. In the example above, if every primitive shape was checked for overlaps *after* both reflections, 11+10+9+...+2+1 = 66 checks would be required. By checking at each stage of reflection, the number of checks is reduced to 3+9+36 = 48.

Moreover, this process of enforcing symmetrical designs illustrates the benefits of the use of correction to prevent illegal phenotypes. Reflecting some designs in a fixed plane will always generate overlapping primitives, whether the positions of individual primitives are specified globally or relative to each other. Because overlapping primitives are already 'squashed' to correct them, the same process can be reused in the symmetry algorithm to ensure that legal symmetrical designs are always generated.

4.3.4 Adding and Removing Primitives

Designs can be modified by repositioning and reshaping the individual primitives that define the designs. However, early experimental work soon showed that although the system could effectively evolve designs from scratch using a sensible (and fixed) number of primitives (determined by trial and error), if too many or too few primitives were specified by the user, the system was unable to evolve acceptable solutions (Bentley & Wakefield, 1995b). Hence, the system needed to have the ability to add new primitives or remove existing primitives from designs automatically.

Unlike the other genetic operators that create and modify designs independently of the representation (described in the next chapter), the two high-level operators that add and remove primitives do act in a manner specific to the solid object representation. These operators manipulate a genotype to remove and add whole primitives from a corresponding phenotype. A primitive is removed by simply deleting it, allowing unnecessary primitives to be discarded from designs. A primitive is added by splitting an existing primitive of a design into two halves, thus allowing greater precision and definition of detail in the design. (If the phenotype is reflected to make it symmetrical, then these operators, by modifying the genotype, will also add or remove all corresponding symmetrical primitives.)

Genetic operators such as these define how the search space can be traversed. If a mutation operator made a radical change to a design (i.e. changing most of the alleles of the genes at the same time, or adding twice as many new primitives), it is almost certain that the fitness of the design would be reduced, not increased. In other words, although mutation should produce random and unexpected changes in solutions, it should not jump too far away from the current point in the search space. Evolution works by a long, slow series of very small changes, not by a few unbelievably lucky jumps (Dawkins, 1976). With this in mind, the two high-level mutation operators were designed to minimise the changes made to designs.

The 'split primitive' operator randomly picks a primitive in the genotype, and splits it into two, preserving the shape, size and mass of the design. The intersecting plane of the original primitive is 'inherited' by the new primitive, see fig. 4.14.

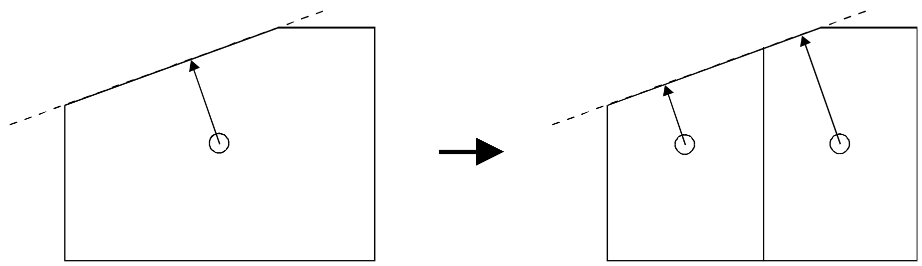


Fig. 4.14 Splitting an existing primitive into two.

To achieve this high-level operation, first a random direction is chosen for the split (parallel to the plane x = 0, y = 0 or z = 0). The original primitive then has its alleles of genes for width halved and x-position changed (if the direction parallel to x = 0 was chosen), with position and size genes for a new primitive being created and given appropriate values. Sharing the plane of the original primitive is more complex, however. First, the equation of the intersecting plane for the original primitive must be calculated (as described earlier in this chapter). Since this equation is defined relative to the centre of the primitive, it must be translated by the (x, y, z) position of the primitive to define it relative to the global origin of the design. It can be calculated that to translate a plane: $Ax + By + Cz + D = 0$ by (p, q, r), only the last coefficient D needs to be altered to:

$D^1 = -Ap - Bq - Cr + D.$

Both the primitives share the two angles defining the orientation of the intersecting plane of the original primitive. However, since the centre positions of both are different from the original, the *planedist* gene must be calculated for both. This is performed by calculating the distance from each centre point to the plane: $Ax + By + Cz + D^1 = 0$, using the equation given earlier in Section 4.2.5.

The other high-level mutation operator: 'delete primitive', makes probably the largest change to a design that the later stages of evolution can tolerate. When a primitive is deleted, not only is the shape of the design modified, but its size and mass also change. Originally, this operator was intended to 'merge' two existing primitives into one, thus minimising changes such as size and mass. Surprisingly, however, experimental tests showed that designs with deleted primitives seemed to be explored by evolution with the same frequency as those with merged primitives, so the simpler and quicker deletion method was chosen. (This can be contrasted with tests performed comparing the addition of a new, random primitive, with the 'split' operator, in which only designs produced by the latter method were ever successful.) The delete primitive operator simply picks a random primitive in the genotype and removes the corresponding nine coded parameters (see Chapter Five).

4.3.5 Special Features of Phenotypes

To aid the ability of the system to evolve a wide range of different designs, phenotypes and selected primitives in phenotypes can be defined before evolution by the user as having a number of 'special' properties, see table 4.3. These properties are stored globally, either referencing the same primitives in all designs, or just referencing all primitives in all designs.

Inflexible primitives
Two dimensional
No intersecting planes
X-symmetry, Y-symmetry, Z-symmetry

Table 4.3 Special features of designs.

When a primitive is specified as being *inflexible*, this means that it cannot be squashed during the correction process. This is useful when the user wishes to provide the position and shape of some primitives - by making these pre-defined primitives inflexible, the system will evolve a design around an inflexible 'skeleton'. The same method of correcting overlapping primitives is used, except that when a normal primitive overlaps an inflexible primitive, it is moved and squashed twice as much, see fig. 4.15. If two overlapping primitives are both specified as being inflexible, they are corrected normally.

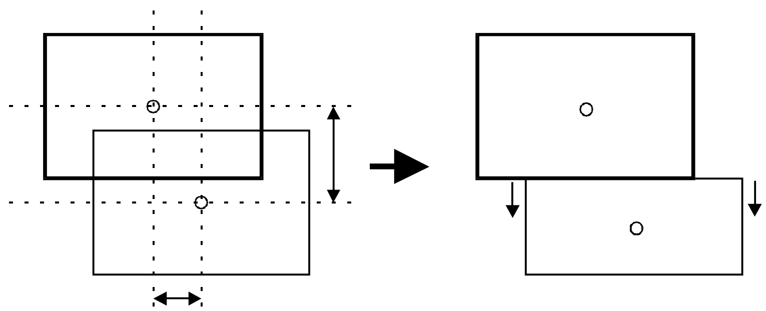


Fig. 4.15 An inflexible primitive is not squashed during the correction process.

Additionally, whole designs can be specified as being two dimensional. All primitives in such designs have a fixed, user-specified value for depth, a fixed z-position value of zero, and a fixed value of zero for one of the angles that specifies the orientation of the intersecting plane (see Chapter Five). In this way the system can evolve in two-dimensions - sometimes very useful in simplifying the design task for the system. The user is also able to specify whether primitives in designs should have intersecting planes or not. Some naturally 'blocky' designs do not require this refinement to the representation. Although the three genes that specify the position of the plane for each primitive do remain in the genotype of such designs, they are ignored when the genotypes are mapped to the phenotypes if this option is selected by the user (see Chapter Six). Finally, as outlined previously in this chapter, designs can be specified as being symmetrical in the planes X = 0, Y = 0, or Z = 0, or in any combination of these planes.

4.4 Summary

The evolutionary design system relies on a suitable generic solid object representation to allow the definition of a wide range of designs, and to enumerate the design-space in a manner conducive to evolutionary search. Out of the existing representations examined, spatial partitioning seems to be the most suitable type of solid object representation. However, traditional spatial partitioning representations often require many parameters and often approximate curved surfaces poorly. To overcome these limitations, a new, low-parameter spatial partitioning representation was developed, capable of approximating curved surfaces more precisely (Bentley & Wakefield, 1996b).

In addition, a simple form of artificial embryology was developed, allowing genotypes to remain unconstrained, but ensuring that all genotypes can be mapped to legal phenotypes (i.e. designs without overlapping primitives). This efficient and fast correction process can also encourage the proper use of the representation in phenotypes (Bentley & Wakefield, 1995a/1996a). The same process was found to be essential when symmetrical designs were required, allowing any design to be transformed into a symmetrical design by simple reflections in planes (Bentley & Wakefield, 1995b).

The evolutionary design system has the ability to add and remove primitives from designs during evolution. High-level genetic operators are used to achieve this, designed to minimise the damage performed by random mutation of this nature by splitting existing primitives into two, or simply deleting existing primitives. Finally, the system allows the user to specify special features in phenotypes to enable evolution around inflexible 'skeletons', two-dimensional evolution, evolution of symmetrical designs, and the evolution of designs defined by primitives without intersecting planes.