

Generic Representation of Solid-Object Geometry for Genetic Search

PETER J BENTLEY & JONATHAN P WAKEFIELD¹

Abstract: *This paper examines the first stage of using genetic algorithms in design - how to encode designs as chromosomes. Generic representations capable of describing the geometry of any solid object to eliminate the unnecessary duplication of work at this stage are explored. A suitable low-parameter spatial-partitioning representation is then proposed, using partitions consisting of six-sided polyhedra intersected by planes. Finally, the coding of the representation is examined, with the conclusion that allelic coding with control genes and user-definable fixed-value alleles would provide the most flexible system.*

1 INTRODUCTION

The genetic algorithm (GA) is rapidly growing in popularity as more researchers discover its 'human-like' search abilities. Although no theory exists to explain adequately why the GA is so versatile, the growing literature in an increasing variety of fields shows that its uses are still being discovered (5).

One such area for which the use of these evolutionary search techniques have become more widespread in the past few years is design optimisation (11). GAs today are being used to optimise designs from the shape of turbine blades to floorplans of oil-rigs (8). However, for

every application of the genetic algorithm, the same three problems must be solved:

- *A fitness function (objective function) must be created*

It is the fitness function that specifies what is a good design and what is not. In a very real sense, the fitness function is the design specification, complete with all constraints. (It has been found that penalising potential design solutions for conflicting constraints with the fitness function is more effective than preventing any designs that conflict the constraints from being created in the first place (12). Even a bad design can have some good features.)

- *The type of GA must be determined.*

Although the traditional 'canonical GA' is adequate for most search problems, for problems involving high numbers of parameters and constraints, some optimised GA variants are required. Examples include: the steady-state GA, the distributed GA (16), the messy GA (6), the structured GA (11), the parallel GA (10) and the Memetic algorithm (14). More advanced genetic techniques and operators can also be employed to improve evolution speed, such as: population overlapping, population seeding, species formation, parasitism, inversion, duplication, deletion, sexual determination, segregation and translocation (5),(15). Picking a good GA and effective operators for a search problem is still something of an art.

¹ P.J. Bentley B.Sc. (Hons) and J.P. Wakefield Ph.D. M.Sc. B.Sc. (Hons),
Division of Computer and Information Engineering, School of Engineering, University of Huddersfield, Queensgate,
Huddersfield, West Yorkshire HD1 3DH, UK
E-mail: pbentley@aol.com

- *The problem must be coded into chromosomes*

Genetic algorithms do not work with the parameters that define designs directly, but instead manipulate coded forms of these parameters. The choice of coding can be different for every application, but it is an important choice, for it determines the type of genetic algorithm and operators that can be used and therefore partially determines the overall speed of evolution of a good solution. Examples of possible codings are: binary, real, genetic, and allelic coding (5),(14). Also single, multiple and structured chromosomes can be used with varying alphabets (5),(11).

The coding of the chromosomes is normally the first stage when creating an evolutionary search system, and it is this stage that this paper will focus on.

2 MAPPING PHENOTYPES TO GENOTYPES

The coding of a design for genetic algorithm must be carefully specified to allow effective and efficient genetic search to take place. This problem of how to encode a design as a chromosome (or string) of genes (or coded parameters) is the problem of *phenotype to genotype* mapping. However, before any such mapping can be specified, an appropriate phenotype representation and genotype representation must be determined.

The *phenotype* can be thought of as a potential solution to the search problem - for the domain of design, a phenotype is one possible real-world design. The phenotype representation is how that phenotype is described (e.g. Bézier surface patches, collections of cubes). The phenotype representation is also an enumeration of the design-space (or 'space' of all designs), with single phenotypes being points in that space.

The *genotype* is the total genetic coding of the phenotype, and it is this, rather than the phenotype, that is directly manipulated by genetic algorithms (5). Exactly what form the genotype takes is described by the genotype representation. Thus, the genotype (or structure) of a single solution to a search problem is

defined by one or more chromosomes, which are a coding of the phenotype.

Obviously for every different design task, a good solution will be a different phenotype. However, today it is also normally the case that for every different design task, the phenotype representation is different. Representations can vary from a few hand-picked dimensions of a design, to parameters specifying every part of the design. Designs can be represented directly, such as lists of vertices and edges, or indirectly through equations (e.g. Parametric surfaces, or swept surfaces) (9),(13). For every one of these different phenotype representations, there will be a different genotype representation required, and often a different genetic algorithm as well. Whilst this has the advantage that only relevant parameters need be considered, it also has the major disadvantage that no two implementations are compatible, meaning that much duplication of work is required for every new design task to be tackled in this way.

It therefore seems appropriate to attempt to create a generic phenotype representation that can be used for multiple design tasks without any need to redefine the genotype representation and phenotype to genotype mapping. Such a generic representation could be used with GAs in three ways. Firstly, an existing design could be automatically represented and then be ready for immediate optimisation by a GA. Secondly, different preliminary designs could be described by the representation and used to seed the initial population of a GA. This would allow evolution to incorporate the best features from these initial designs into the final evolved design. Thirdly, by seeding the initial population with random designs, the GA could evolve potentially new conceptual designs.

In other words, the aim of this work is to attempt to create an enumeration of the whole design-space, in which the solution to any suitably specified design task can be found. For the purposes of this paper, the 'design-space' will be thought of as the infinite space containing every possible design of solid objects. Hence, this paper will concentrate on the creation of a representation capable of defining any solid object, with the assumption that any 3D representation can also represent 2D designs. This work ignores factors such as material, cost

and surface appearance (i.e. colour), focusing instead on the most difficult aspect of design representation - the geometry of designs.

3 PHENOTYPE REPRESENTATIONS

Using adaptive 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. (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.) So the phenotype representation must be capable of adequately defining a shape using the minimum 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 should ideally allow any design to be changed in any way, to any degree. Evolution demands small, gradual changes, enumeration that places very dissimilar designs 'side-by-side' in the space would be more difficult to traverse.

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 previously mentioned, the representation must be able to define adequately any 3D shape, so must 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.

There are two main types of suitable 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 (4).

Surface representations (or boundary representations), 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 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. Examples of surface representations include: Polygon Mesh, Quadric, Hermite, Fourier (4), Parametric (7), Bézier, B-Spline (1).

Solid representations are perhaps a better choice for the purposes of evolutionary search, for they can not only define solids with good accuracy and few parameters, but also surfaces, 2D and even 1D shapes. There are three commonly used types of solid representation: sweeping, Constructive Solid Geometry (CSG), and spatial partitioning. Sweeping is not suitable for complex designs - combining swept objects is normally only performed after converting to an alternative representation (4). CSG is one of the most commonly used representations in CAD packages today (3), combining 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.

However, spatial partitioning representations seem ideal. They all involve decomposing the 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 (4).

The advantages of spatial partitioning when compared to other representations are clear. It is the easiest 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. 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.

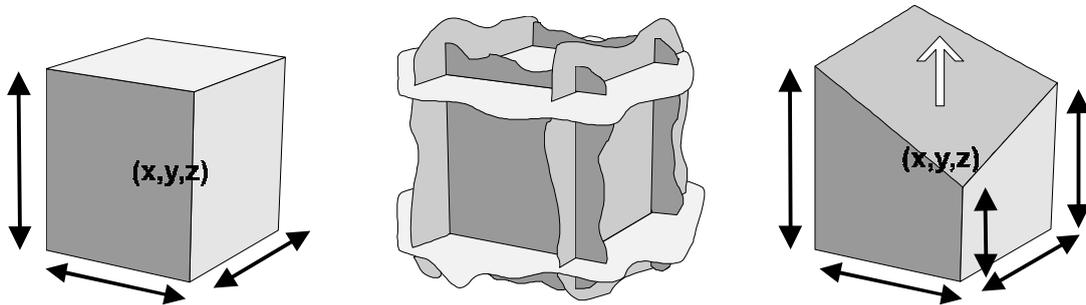


Figure 1 From left to right:
stretched cube, six-sided polyhedron using planes, stretched cube with moveable side

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), the representations are 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 'growth' of new features in designs.

4 LOW-PARAMETER SPATIAL PARTITIONING REPRESENTATIONS

The form of spatial partitioning to be used is essential to ensure the fewest parameters possible. 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 prevent mutation from changing one primitive into another and thus creating a radically different design. The most obvious type of partitioning, using just one type of primitive immediately springs to mind:

4.1 Stretched Cubes: (6 Parameter)

This representation is a variant of octrees (4), consisting of cubes with alterable lengths, widths and depths, the solid being composed of a number of such stretched cubes, adjacent to each other and non-intersecting. Every such polyhedron would require 6 parameters (x , y , z , *width*, *depth*, *height*), see Fig. 1 (left).

The major disadvantage of this method of representing solids is that huge numbers of the primitives would be required to approximate

curved surfaces or planes not parallel to the sides of the primitives. All solids except the most cube-like would be full of 'jaggies' giving a very stepped appearance and would probably take considerable time to generate. With so many small primitives required for the better approximations of solids, the number of parameters and the corresponding search-space would also become enormous, see Fig 2.

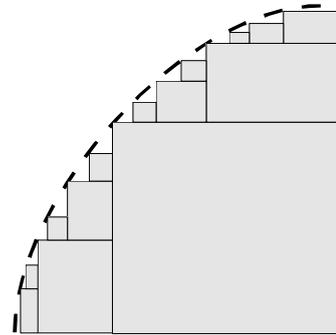


Figure 2 Poor approximation of curve even with large numbers of 'stretched cubes'.

4.2 Six-sided Polyhedra 1: Vertices

An alternative is to allow the six-sided polyhedron primitive to be defined by its eight corner vertices. By 'tweaking' any vertex, the orientation of the three sides it helps to define could be modified, although the vertices making up the sides would have to be kept planar at all times. To prevent a primitive conflicting with its neighbours when an adjacent side is modified, only external sides of the primitives could be made moveable.

The major advantage is that curves and planes of any orientation could be represented much more accurately with far fewer primitives than the 'stretched cube' representation described previously. The major disadvantage of this

method is the parameter-count, which increases to 24 (8 (x, y, z) points) for every primitive.

The parameter count could be reduced by sharing vertices amongst adjacent primitives, i.e. two primitives side-by-side only require 4 parameters to define their shared side (as long as the sides are identical). However, this would not have a great effect if primitives of widely differing sizes were used and that is likely to be the case.

4.3 Six-sided Polyhedra 2: Intersecting Planes

An alternative way to represent such six-sided polyhedra is instead of storing the positions of the vertices, to store the positions of the faces of the cubes. In other words, every primitive with 8 vertices has only 6 sides, and since a side can be defined by a plane needing just three parameters, a 6-sided solid can be fully defined by 18 parameters, see Fig. 1 (middle). Although planes on their own are infinite, when two planes intersect, a line is defined; when six intersect, a solid can be defined. (This method of defining solids is often used in ray-tracing (4).) The order of the parameters in any coding would be vital, for the 'inside' of the solid would need to be defined.

The advantage of such a representation is that the primitives would have exactly the same flexibility and capabilities of the 'vertices' method, with 6 fewer defining parameters per primitive. Perhaps the biggest disadvantage is that in some situations, the representation could be ambiguous, for six intersecting planes can define more than one solid at a time. Also, 18 parameters per primitive is still very high.

The parameter-count and effective search-space could potentially be reduced by sharing planes that adjacent primitives have in common. This would be more beneficial than sharing of vertices, for planes are always shared by touching primitives whilst vertices are not. It also seems like an elegant solution, for it would allow actual definition of 'touching' in the representation. For example, consider a table with four legs, made from 5 primitives. The bottom of the table shares the same plane as the tops of all four legs since they are touching, meaning a reduction from 90 parameters to 78.

The problems arise with the referencing mechanism for determining which solid has which planes - more parameters are required. Although they would not be treated in the same way when searching the subset of the design-space, they are alterable (a primitive may mutate to a non-touching position with another primitive) and so do increase the effective parameter count. It is possible that large quantities of touching primitives could make up for this slight increase, but, despite its seeming elegance, this does not appear to be the most efficient representation.

From the investigation so far, it is clear that whilst the six-sided polyhedron (a polyhedron that can have any of its sides oriented in almost any direction) representations give far greater solid definition, they all have the problem of massively increased numbers of parameters. The only representation with a good, low number is the 'stretched cube' representation - but as explained earlier, these are very limited in their accuracy and so huge numbers of primitives are required to define any complex object. The logical step is to combine the two types of representation and attempt to get the best of both worlds: few parameters and flexible definition capabilities.

4.4 Stretched Cubes with Single Moveable Sides

This can be achieved by allowing 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 should be possible to use it to approximate a surface of any orientation. The primitive would need 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. 1 (right).

Although this seems to be an ideal representation of solids, there are in fact two major disadvantages. 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. This can be solved by

using 'stretched cubes' for the inside of a solid and stretched cubes with moveable sides for the outside. 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. 3.

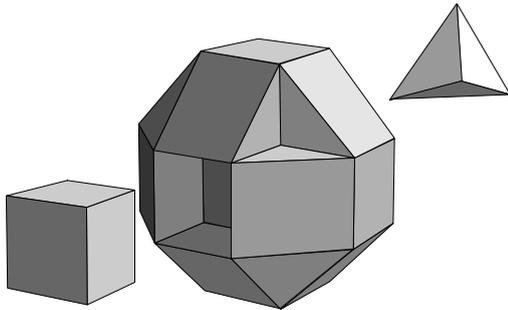


Figure 3 Four-sided polyhedron with triangular sides cannot be formed by primitive - representation fails.

4.5 Stretched Cubes and Intersecting Planes

By examining the type of primitive required to represent varying curved surfaces (e.g. see top right of Fig. 3), it becomes clear that the primitive required 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 the primitive, no orientation parameter is required, see Fig. 4.

This representation can approximate closely any curved surface, and with few parameters required per primitive, it is the most compact and flexible of all of the representations examined so far.

5 MAPPING THE SPATIAL PARTITIONING REPRESENTATION: CLIPPED STRETCHED CUBES

As described above, 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. 5. 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.

As the reference point (x, y, z) defines the centre of the primitive, the eight vertices of the stretched cube are simply:
 $(x \pm width, y \pm height, z \pm depth)$

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: $Ax + By + Cz + D = 0$ (4)

the plane coefficients can be calculated as:

$$A = d \cos\beta \sin\alpha$$

$$B = d \sin\beta$$

$$C = d \cos\beta \cos\alpha$$

$$D = -(A^2 + B^2 + C^2)$$

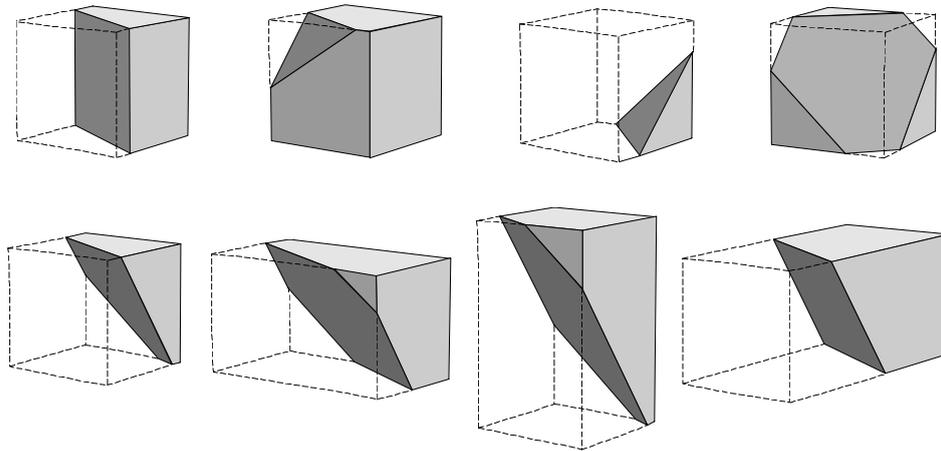


Figure 4 Clipped stretched cubes

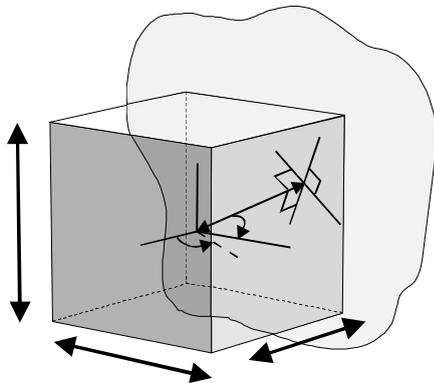


Figure 5 Clipped Stretched Cube: ($x, y, z,$
width, height, depth, planedist, angle1, angle2)

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 12 edges between the 8 existing vertices of the stretched cube.

So, given an edge defined by 2 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: $Ax + By + Cz + D = 0$

Point: $P_1(x, y, z)$ is 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:

$$x_2 \geq x \geq x_1$$

$$y_2 \geq y \geq y_1$$

$$z_2 \geq z \geq z_1$$

However, when implementing the above, the following test should be used instead:

$$(x_2 + \text{MATHSERR}) > x > (x_1 - \text{MATHSERR})$$

$$(y_2 + \text{MATHSERR}) > y > (y_1 - \text{MATHSERR})$$

$$(z_2 + \text{MATHSERR}) > z > (z_1 - \text{MATHSERR})$$

where $\text{MATHSERR} = 1.0\text{E-}7$

The MATHSERR constant is required otherwise the small errors produced by the binary coding of real numbers would prevent correct results. (Alternatively, line-clipping algorithms such as Cohen-Sutherland or Cyrus-Beck can be used (4).)

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, 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:

$$Dist = \frac{Ar + Bs + Ct + D}{\sqrt{A^2 + B^2 + C^2}}$$

However, since only the sign of the distance is required, using:

$$Dist = Ar + Bs + Ct + D$$

is sufficient.

The remaining vertices are the corners of the primitive, and define the size and orientation of its sides. The algorithm for mapping a 'clipped stretched cube' primitive coded within the genotype to a polyhedron in the phenotype is thus:

Extract the 9 primitive definition parameter values from the chromosome
(see next section)

Use ($x, y, z, width, height, depth$) to generate the 8 vertices;
(the 8 vertices define 6 sides and 12 edges)

Calculate the new vertices (if any) generated by the plane ($angle1, angle2, distance$) intersecting the 12 edges.

Calculate the distance of each vertex from the plane, if the value is positive, remove that vertex. (Remove any vertices that are on the outside of the plane.)

Divide vertices into groups of coplanar vertices - each group defines a side
(each vertex will be shared amongst three different groups)

Sort the vertices for each side into the correct order for displaying
(i.e. all vertices of a side must be connected by edges, but no 2 edges should intersect)

Once the 4-7 sides (depending on the plane) of the primitive are generated, it can be displayed and additional information such as surface area and volume calculated.

Hence, the set of coded values in a chromosome is mapped to the shape of a solid object by the division of that shape into a collection of more primitive shapes (stretched cubes and clipped stretched cubes), which are described in terms of a list of parameter values. These values are coded in a chromosome using the method defined by the genotype representation.

6 GENOTYPE REPRESENTATIONS

An artificial chromosome within a genetic algorithm can take many forms, from the variable length trees of genetic programming to the fixed length strings of the canonical GA (5). There are two main categories of coding used to define the genotypes within chromosomes: genetic coding and allelic coding (14); both can be either unstructured or structured (11).

The simplest approach is genetic coding, where genes correspond directly to parameters (every gene is a coded parameter) and are stored in a list or string of fixed length and order (the chromosome) (5). Normally the order of the genes determines the mapping relation, e.g. the sixth gene could correspond to 'depth of primitive 1'. Such a chromosome can be made into a 'structured chromosome' as used in the structured GA by the addition of 'control genes' which switch on or off other genes in the chromosome to add or remove them from the genetic search (11). As long as the number of parameters in every phenotype remains constant then the numbers of genes in every genotype will remain constant, providing no problems during reproduction. In addition, as long as a designer seeds the initial population of the GA with designs containing enough primitives to allow a better design to evolve, the genetic coding will be sufficient. However, should more (or less) primitives be required to represent a good design fully than were initially specified, the fixed-

length chromosomes of genetic coding will present problems.

Problems arise in situations where it is desirable to use mutation to add a new primitive to a design. This would result in one chromosome containing more genes than all of the others. When performing the genetic operation of crossover upon chromosomes of differing lengths during reproduction, reliable results cannot be provided. For example: standard single-point crossover produces a new chromosome based on segments of two parent chromosomes. If the parents were 'abcdefg' and 'ABCDEFGF', with a random crossover point of, say 5, the offspring would be: 'abcdeFG'. However, if the first parent had been mutated into a longer length such as 'abcxyzdefg', the resulting offspring would become: 'abcxyFG', with two old genes 'de' and one new gene 'z' being lost. When decoded, this would be a phenotype with parameters missing and would be meaningless.

A more recent idea of Radcliffe (14) is to use allelic coding to solve this problem. Using this form of coding, chromosomes become more like sets than lists, with alleles within them instead of genes. An allele is the value a gene can take, with every allele belonging to one and only one gene. Alleles are thus independent of any locus (or position) within a chromosome. Storing locus-independent coded parameter values would prevent any problems of the locus (or position) of genes being disrupted. In other words, instead of storing a list of values whose position in the list determines their use, an unordered collection of values is stored, with every value 'knowing' which parameter it belongs to. An example of a possible allelic coding of a solid object could be:

```
[ Primitive1_width_24.8
  Primitive20_height_12.32
  Primitive1_xpos_101.0 ...
Primitive20_angle1_0.34221
  Primitive1_planedist_5.2 ]
```

Reproduction can take place with any of a variety of crossover operators that exist for this form of coding (14), with the advantage that, should it be required, primitives can be added or removed from any phenotype. An additional advantage is that the chromosome does not even

have to contain all the alleles for each primitive. Should, for instance, the value of the width parameter be missing for a primitive, a default value could be automatically used instead, allowing a further reduction of the effective search space. The coding can also be extended to resemble the structured genetic coding, with the addition of control alleles to switch on or off selected groups of alleles. An alternative or addition to using control alleles is also to allow a designer to specify which alleles are to remain unchanged throughout the searching process. These can then be fixed in value by the program, allowing only the part of the design that the designer feels should be optimised to be evolved using the GA. In this way, the generic representation can be used without any unnecessary extra parameters ever being examined by the GA.

Perhaps the biggest disadvantage of this form of coding is the fact that every value must have an identifier attached to it. If the identifiers themselves were made variable, this would increase the effective parameter-count and hence the search-space, but keeping them fixed will prevent this. In other words, the value 'Primitive1_width_23' should be prevented from mutating to 'Primitive1_depth_23' or even 'Primitive2_width_23'.

Finally, the coding of the parameters themselves must be considered. The usual coding is either binary or real (base 2 or base 10). It has been found that for parameters taking a large range of values, the GA searches most quickly when real coding is used (8). Using binary does improve the convergence of a GA to a good solution, however.

Consequently, perhaps the best form of genotype representation to use is allelic coding, with alleles being stored as real numbers (floating-point in base 10). Provision should be made for selected parameters to be fixed in value and the addition of control alleles. Thus, with these representations and the codings combined, a GA would be able to evolve the dimensions, positions and number of primitives that make up a design. In this way, the precise geometry of the design could be evolved, as specified by design evaluation software (fitness function) (2).

7 CONCLUSIONS

When applying a genetic algorithm to any search domain, the first step is normally to encode the search problem. Often the least thought goes into this area of the creation of a new GA application, resulting in the creation of countless incompatible codings. By the creation of a single, generic phenotype and genotype representation for the domain of design of solid objects, very different designs can be optimised using GAs, without the unnecessary duplication of work.

The spatial-partitioning solid representation 'clipped stretched cubes' introduced in this paper allows any solid to be approximated very closely with a small number of definition parameters. The corresponding coding of this representation is therefore very compact, meaning the search-space examined by the genetic algorithm is small, thus reducing the combinatorics and improving the overall speed of searching. By using a very low-parameter spatial-partitioning representation, all design tasks involving the optimisation or creation of three-dimensional solid shapes can be coded immediately. Indeed, with an automatic conversion from the standard CAD format of CSG to the 'clipped stretched cubes' representation, any such design can be automatically coded ready for optimisation using GAs instantly.

The allelic coding of the 'clipped stretched cubes' representation allows any phenotype to be refined by the optimisation of not only the dimensions of the primitive shapes it is made from, but by the optimisation of the quantity of the primitives. In other words, not only can the GA optimise the geometries of existing primitives in a design, it can add or remove primitives to provide more or less detail respectively, in parts of the design. In this way the GA can itself define the optimal effective search space that it needs to explore. The search time can also be reduced by allowing the designer to 'fix' the values of any parameters that specify parts of the design which are considered good enough, thus allowing a generic representation to retain the benefits of an application-specific representation.

This representation could be used with a GA in three ways. Firstly, the geometry of an existing design partitioned using the 'Clipped Stretched Cubes' representation could be simply and easily optimised by a GA. Secondly, by seeding the initial population of the GA with a number of different preliminary designs, some even defined by different numbers of primitives, a new design could be evolved incorporating the best features from the preliminary designs. Thirdly, the geometry of designs could be evolved from scratch (i.e. random seeding of the initial population) to provide potentially new conceptual designs.

The purpose of this work was to produce a generic representation capable of representing the geometry of a wide range of designs in a manner suitable for evolution by genetic algorithms. Work in progress will address the application of the representation to a range of design tasks. Examples of suitable design optimisation/creation tasks include: tables, fences, optical prisms, bridges, gear teeth and a variety of broader problems such as floorplanning and packing. Early work using the representation with a simple genetic algorithm to evolve designs from scratch have provided some very successful and novel results (2).

8 REFERENCES

1. Anwei, L., Xin, Y., Hua L., Shenquen, L., "HYBRID -- A Solid and Surface Modeling System Based on Database", CAD and Computer Graphics '89, Beijing, 1989, pp. 222-227.
2. Bentley, P. J. and Wakefield, J. P., "The Evolution of Solid Object Designs using Genetic Algorithms", Applied Decision Technologies, London, 1995, pp. 391-400.
3. Chuan Jun Su, Mayer, R. J., Browne, D.C., "Generalised CSG: A Solid Modeling Basis for High Productivity CAD Systems", Autofact '91, 1991, pp. 17/35-17/43.
4. Foley, J., van Dam, A., Feiner, S., Hughes, J., *Computer Graphics Principles and Practice*, 2nd ed., Addison-Wesley, 1990.
5. Goldberg, D. E., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 1989.
6. Goldberg, D. E., Deb, K., Kargupta, H., Harik, G., "Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms", Illinois Genetic Algorithms Laboratory (IlliGAL), report no. 93004, 1993.
7. Joy, K. I., "Utilizing Parametric Hyperpatch Methods for Modeling and Display of Free-Form Solids", Symposium on Solid Modeling Foundations and CAD/CAM Applications, 1991, Rossingnac, J. & Turner, J. (eds), pp. 245-254.
8. Keane, A. J., "Experiences with Optimizers in Structural Design", Adaptive Computing in Engineering Design and Control '94, Plymouth, 1994, pp. 14-27.
9. King, E. G., Freeman, L. M., Karr, C. L., Whitaker, K. W., "The Use of a Genetic Algorithm for Minimum Length Nozzle Design: A Process Overview", Third Workshop on Neural Networks: Academic/Industrial/NASA/Defense, 1993, pp. 556-563.
10. Levine, D., "A Parallel Genetic Algorithm for the Set Partitioning Problem", D. Phil dissertation, Argonne National Laboratory, Illinois, USA, 1994.
11. Parmee, I. C. and Denham, M. J., "The Integration of Adaptive Search Techniques with Current Engineering Design Practice", Adaptive Computing in Engineering Design and Control '94, Plymouth, 1994, pp. 1-13.
12. Parmee, I. C. and Purchase, G., "The Development of a Directed Genetic Search Technique for Heavily Constrained Design Spaces", Adaptive Computing in Engineering Design and Control '94, Plymouth, 1994, pp. 97-108.
13. Pham, D. T. and Yang, Y., "A Genetic Algorithm based Preliminary Design System", Journal of Automobile Engineers, Vol 207, No.2, 1993, pp. 127-133.
14. Radcliffe, N. J. and Surry, P. D., "Formal Memetic Algorithms", Edinburgh Parallel Computing Centre, 1994.
15. Robbins, P., "The Effect of Parasitism on the Evolution of a Communication Protocol", The 3rd International Conference on Simulation of Adaptive Behaviour (From Animals to Animats 3), Brighton, 1994, pp. 431-437.
16. Whitley, D. and Starkweather, T., "GENITOR II: a distributed genetic algorithm", Journal of Experimental and Theoretic Artificial Intelligence, Vol 2, No. 3, 1990, pp. 189-214.