Reinterpreting Rasmi Domes with Geometric Constraints: A Case of Goal-seeking in Parametric Systems

Maryam M. Maleki and Robert F. Woodbury

Reinterpreting Rasmi Domes with Geometric Constraints: A Case of Goal-seeking in Parametric Systems

Maryam M. Maleki and Robert F. Woodbury

Geometry has long been a generator of architecture. In traditional Persian architecture, Rasmi domes project a drawing onto a predefined 3D geometry. In fact, the word 'rasmi' and the verb for drawing in Persian have the same linguistic root. Projection is readily done in manual drawings or conventional CAD programs. From a constraint perspective, the dome is constrained by the drawing and the 3D geometry. If the latter constraint is replaced by invariance of distance on the original drawing, a class of domes results, but members of this class cannot be computed conventionally. Class members are developable from a planar layout of triangles, which is, in turn, generated by a simple drawing rule. This yields a parametric structure of four parameters. Three determine the initial planar diagram. One determines configuration. Further, domes in the class are mechanisms: they are not fully specified by the constraints and parameters. We develop the geometric constraints representing the location of the defining points of a dome and present a goal-seeking algorithm to solve the constraints within a propagation-based parametric modeling system.

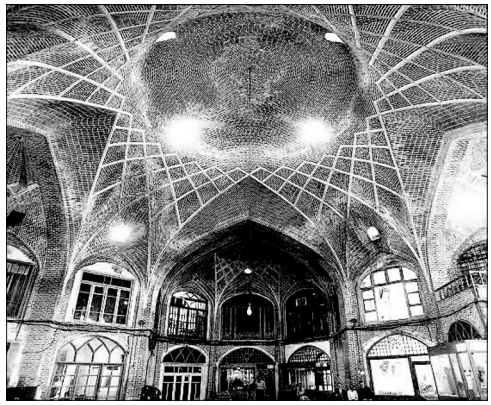
1. RASMI DOMES IN PERSIAN ARCHITECTURE

In the design of traditional buildings in Iran, we find many examples of the use of geometry as a generator of architecture, in both structural elements and decorative forms and patterns. Distinct form families arise from simple geometric constraints, as in *Rasmi* domes. "Rasmi" in Persian refers to something that comes from a drawing, and a Rasmi dome is a type of dome with specific decorative pattern under the dome.

Figure 1 shows how Rasmi domes are used singly and in combination to define centrally focused or axial spaces. Single Rasmi's are often used in large domes in bazaars, with or without an opening in the dome centre, and in the main domes of mosques. A series of Rasmi's are also used in smaller domes to cover the large prayer areas in mosques.

Nimkar is a form of Rasmi that consists of a semi-dome (see Figure 2). Nimkars typically cover entrances or balconies.

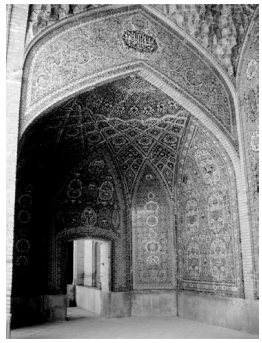
► Figure 1. Examples of Rasmi domes.



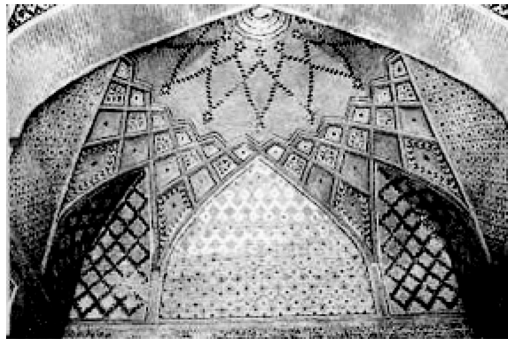
(a) Rasmi dome in bazaar, Kashan, Iran

(b) A series of Rasmis in Nasir-Al-Molk Mosque, Shiraz, Iran

► Figure 2. Examples of Nimkars.

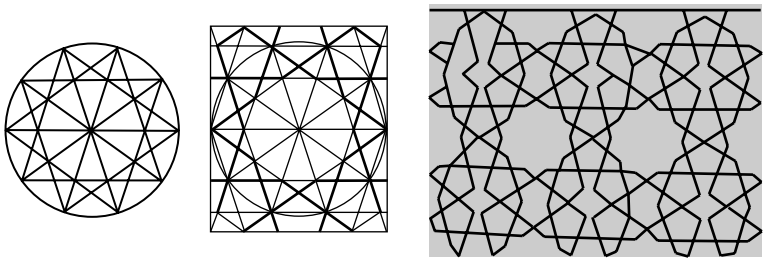


(a) Nimkar in an entrance, Nasir-Al-Molk Mosque, Shiraz, Iran



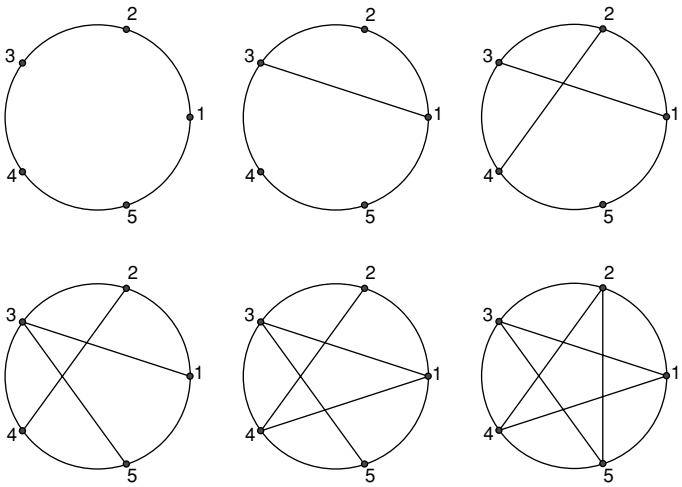
(b) Nimkar in a balcony, Jame Mosque, Isfahan, Iran

A Rasmi dome is generated by selecting key points of a diagram called a *star pattern*. Star patterns have been used by Islamic architects in several ways to create decorative patterns, intact as a pattern themselves, or as a base for other, more complex patterns. (see Figure 3)



◀ Figure 3. Analysis of a pattern in GazurGah, Herat, Afghanistan [2].

Figure 4 shows that a star pattern can be created by evenly putting n points on a circle and drawing *connecting lines* from each point to the d^{th} next point on that circle. We call n the number of *points* in a star and d the *jump*. The resulting pattern is referred to as a star of n/d. Star patterns can be found in Islamic art and architecture in which $2 \leq d \leq n/2$[3][1] . When $d = 1$, the result is a n-gon. For a star pattern to be *used in* the creation of a dome, $3 < d < n/2$.



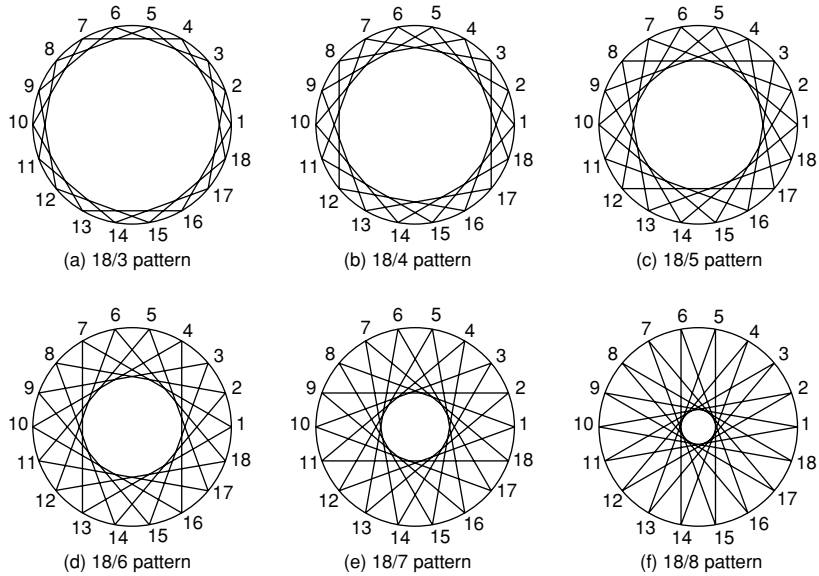
▶ Figure 4. Sequential construction of a star pattern of 5/2.

Figure 5 shows 18-point star patterns with jumps of 3 to 8 points. The opening in the middle becomes smaller as d increases. This clearly has implications for the utility of a particular star pattern as a generator of architectural enclosures.

Connecting the points on the circle in a star pattern results in the creation of new sets of points on the intersection of each line with the other lines. The points on the circle are called the *first row* of points, as they

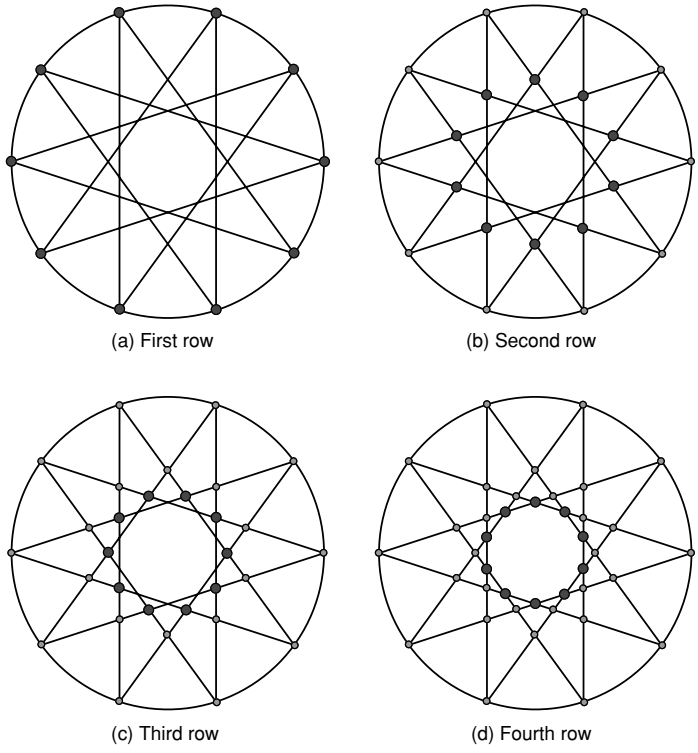
[1]Kaplan specifies $2 < d < n/2$, which error we correct here.

► Figure 5. Possible star patterns with 18 points on the circle.

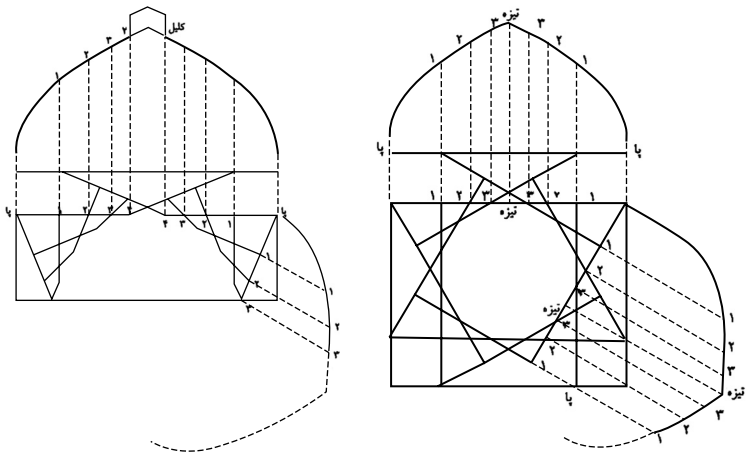


are the first points that the lines intersect. Counting from outside in, the second intersections of each two lines are collectively called the *second row* of points, the third intersections the *third row* and so on, as shown in figure 6. Each row of points has *n* members.

► Figure 6. Rows of points in a star of 10/4.



Representing the traditional Rasmi dome requires selection of intersection points on a star pattern and projection onto a dome section taken over the chords of the Rasmi dome diagram. Figure 7 shows modified blueprints drawn by traditional Persian architects to describe ways of creating Rasmi domes for their students. After projecting the points on the wooden model of the dome section, they would use that model to complete a dome [4].



(a) Nimkar (b) Rasmi

◀ Figure 7. Projecting the 2D pattern on the dome [4].

In figure 8 we represent the same diagram in 3-dimensions, and present the steps of projecting the star pattern to the dome.

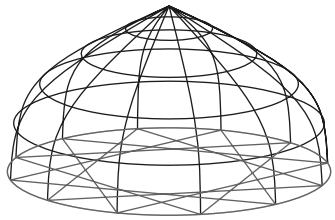
In traditional Rasmis, Persian architects try to keep the shape of the dome fixed. In other words, the shape of the dome is the constraint in transforming a 2D star pattern into a Rasmi dome. But there are other constraints from which related forms can arise.

2. FOLDING A STAR PATTERN TO CREATE A DOME

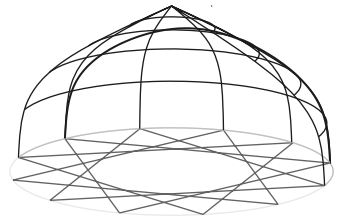
We change the geometric constraints in this model, from the shape of the dome as Persian architects do to the length of the line segments in the star pattern. Figure 9 shows that a simple physical way of maintaining length constraints is drawing the star pattern on a piece of paper and folding it on its lines. In this case, because paper doesn't shrink or stretch, the line segments, and consequently the triangles, stay fixed in shape throughout the 2D to 3D transformation. Of course, this is a special case: the original shape need not be planar for the length constraint to be used.

Here we have a new class of dome structures that are simultaneously coherent with examples from traditional Persian architecture, and with the contemporary concern with kinetic (moveable) architecture. The class has the property that its members are developable from a planar layout of triangles,

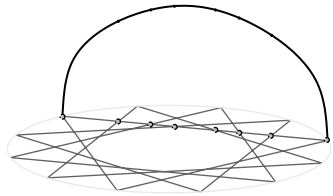
► Figure 8. The process of creating the Persian dome.



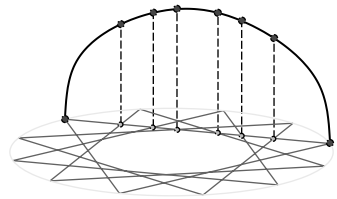
(a) Drawing the 2D star pattern under the dome



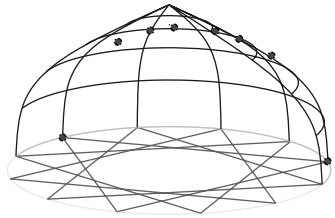
(b) Making a model of the section of the dome on a connecting line



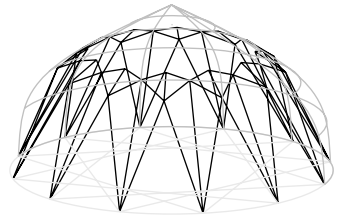
(c) Finding the intersection points on the pattern



(d) Projecting the points to the section model

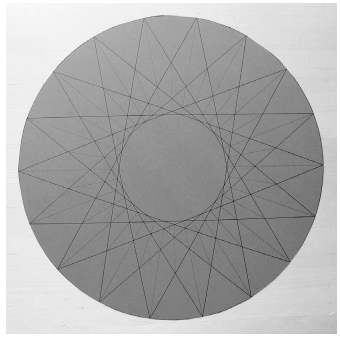


(e) Transferring the points to the dome on all of the connecting lines

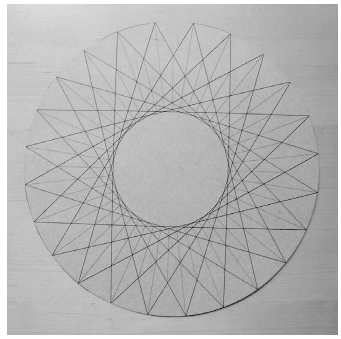


(f) Connecting the points on the dome

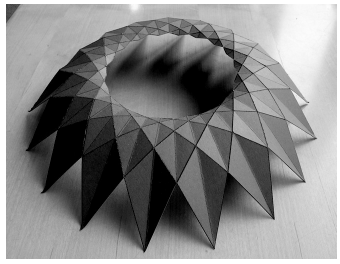
► Figure 9. Folded paper modal.



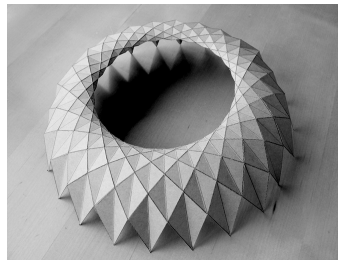
(a) 24/7 pattern



(b) 36/9 pattern



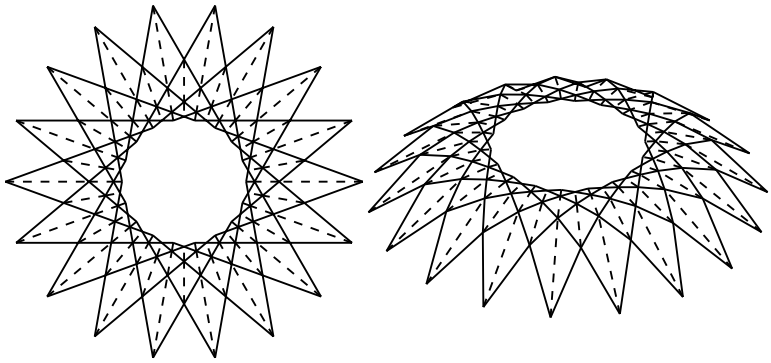
(c) 24/7 dome



(d) 36/9 dome

which are in turn generated by a simple drawing rule. All of its members are also mechanisms, that is, the geometric location of elements is not completely specified by parameters. This class is describable as a parametric structure of four parameters. Three are used to generate the initial planar diagram. One is used to determine configuration. Representing the class requires maintenance of the length constraints given by the initial diagram.

In order to fold a star pattern, we need to remove the middle section of each connecting line. In other words, in a star of *n*/*d*, for each line, the section after its $(d-1)^{th}$ intersection with other lines will be removed. This leaves a non-convex *central polygon* that is free of lines. At this point, the visible lines are the ridge creases. Now we add the valley creases by drawing *radial lines* from each point on the circle to the center of the circle and removing the part in the middle as shown in figure 10. Continuous lines show the ridge creases and dashed lines show the valley creases. To create the paper model, we must remove the central polygon and the outer wedges between the circle and the star pattern.(See Figure 9.)

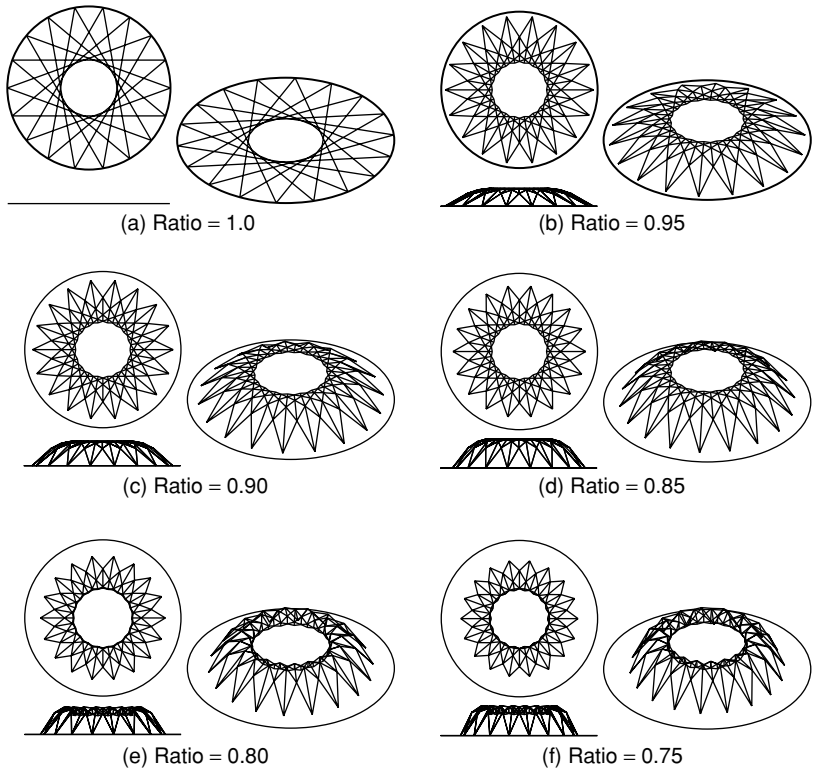


◀ Figure 10. Model before and after folding.

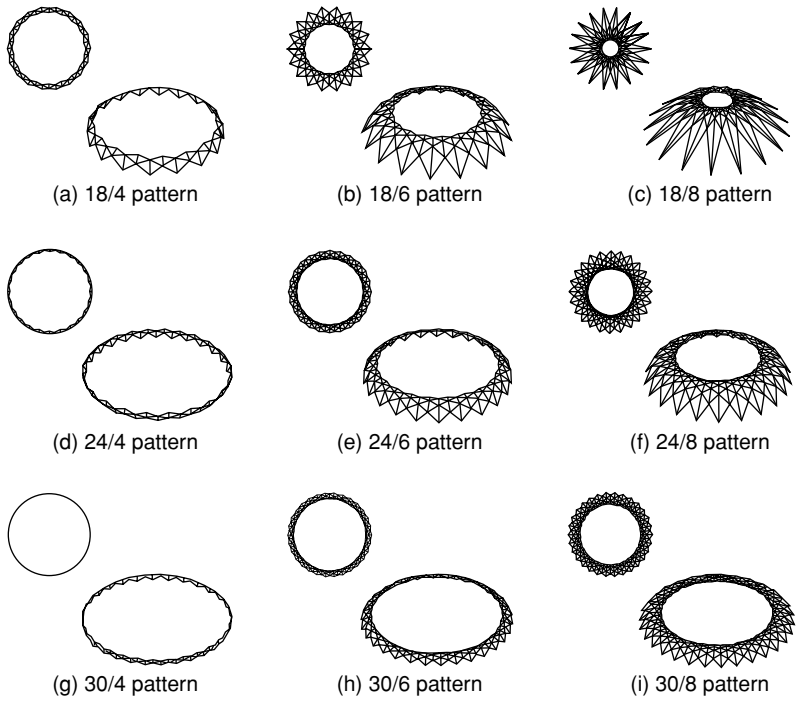
We call the base circle in the 2D star pattern the *2D circle* and its *2D radius R*. After folding the pattern, the circle that the dome covers we call the *final circle* and its *final radius r*. So *r* is always smaller than *R*. When we fold a paper pattern, we can push the final circle more and more inside and it changes the height and shape of the dome. Figure 11 compares domes of *n*/*d* = 18/7, with the same *R* and different *r*s in 3 views. By decreasing the ratio of *r*/*R*, the height of the dome increases. At the same time, the curvature of the dome increases and the dome folds inward. The four dome parameters are thus the number of points *n*, the jump *d*, the 2D radius *R*, and the final radius *r*.

With the same *R* and *r*, the shape of the dome depends on the number of points on the circle, *n*, and the number of jumps, *d*. As shown in figure 12, with the same *n*, the dome closes in the middle when *d* increases. On the other hand, by increasing *n*, the dome will have more and smaller facets.

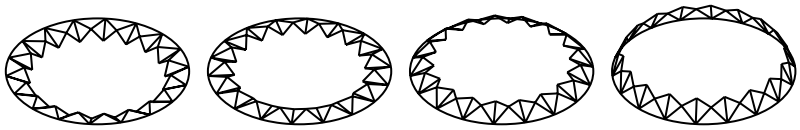
► Figure 11. Domes of 18/7 with the same *R* and different *r*'s.



► Figure 12. Different variations of the dome.



In any star pattern, we can choose to draw a subsequence of the point rows. Figure 13, showing only the first three point rows, makes the structural mechanism clearly apparent. If only two point rows are drawn, the second row of points could, in fact, rotate freely around the first row. Each angle of rotation yields a *physically feasible configuration*. Adding even a single row of points significantly constrains the physically feasible configurations. As the number of point rows grows, the mechanism becomes even more constrained and more difficult to visualize.



◀ Figure 13. Different configurations in the allowable range for the first row of polygons.

3. CREATING THE DOME IN CAD SYSTEMS

The algorithm for creating the traditional Persian dome (Rasmi) in any CAD system is very similar to the traditional construction algorithm: draw the 2D diagram, find the section of the dome on the connecting lines, project the points on the sections, connect the points. It is simple, which makes a lot of sense, since traditional architects had to do it manually, without the computational technology that we have today.

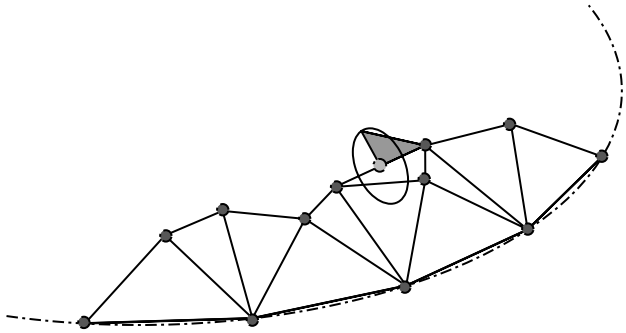
As soon as we change the geometric constraints of the model, it becomes much more difficult, depending on the type of the parametric modeler being used. The class presents challenges to the GUI of available constraint-based systems (for example, SolidWorks). Representing such domes in a propagation-based parametric modeler requires using a *goal seeker* strategy in which the outputs of the model are iteratively recycled to its inputs.

In order to understand the process in any of these systems, we need to take a closer look at the geometry behind this mechanism. (See Figure 14.) Consider that the lengths do not change at any time. Each point P_i lies at a fixed distance along a connecting line from the two nearest points on the $(i-1)^{th}$ row. At the same time, P_i on the i^{th} row of the dome lies at a fixed distance from the corresponding point P_2 on the $(i-2)^{th}$ row along their shared radial line. Thus P_i lies at the intersection of three spheres, centred on the two nearest points on the $(i-1)^{th}$ row and on the corresponding point on the $(i-2)^{th}$ row respectively.

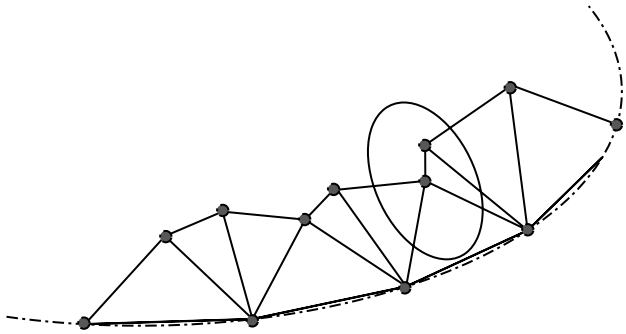
The underlying star patterns guarantees that the two connecting lines from the nearest points on the $(i-1)^{th}$ row are of equal length. Presuming that the dome configuration is radially symmetrical (this is not a physically necessary constraint, but suffices for this process of finding a feasible dome configuration), the two spheres centred on the nearest points on the $(i-1)^{th}$ row are also radially symmetrical and this forces their intersection to be on

a vertical plane running through the dome centre and P_i. Using this plane to cut the circle centred on P_2 yields two coplanar circles, which constrain where P_1 lies. Therefore, to find a point on the i^{th} row, we need to have the points on the previous rows, then draw the corresponding circles on the *radial plane* (the vertical plane containing the radial line), and find their intersections. A configuration is feasible exactly when these two circles intersect. We choose the top point between the two intersections for a dome that goes upward and the bottom one for a dome that goes down, but the same choice for all of the rows.

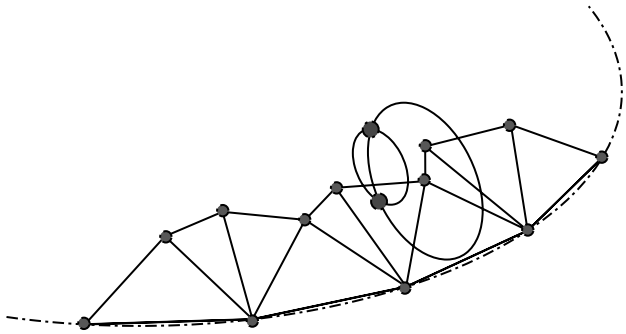
► Figure 14. Finding a point on the fourth row of a dome. The diagram is without loss of generality – it applies to finding any P_i where $i \geq 2$.



(a) The first constraint is P_i must lie equidistant from the nearest points on the $(i-1)^{th}$ point row and at distance equal to the length of the connecting lines between P_i and these points.

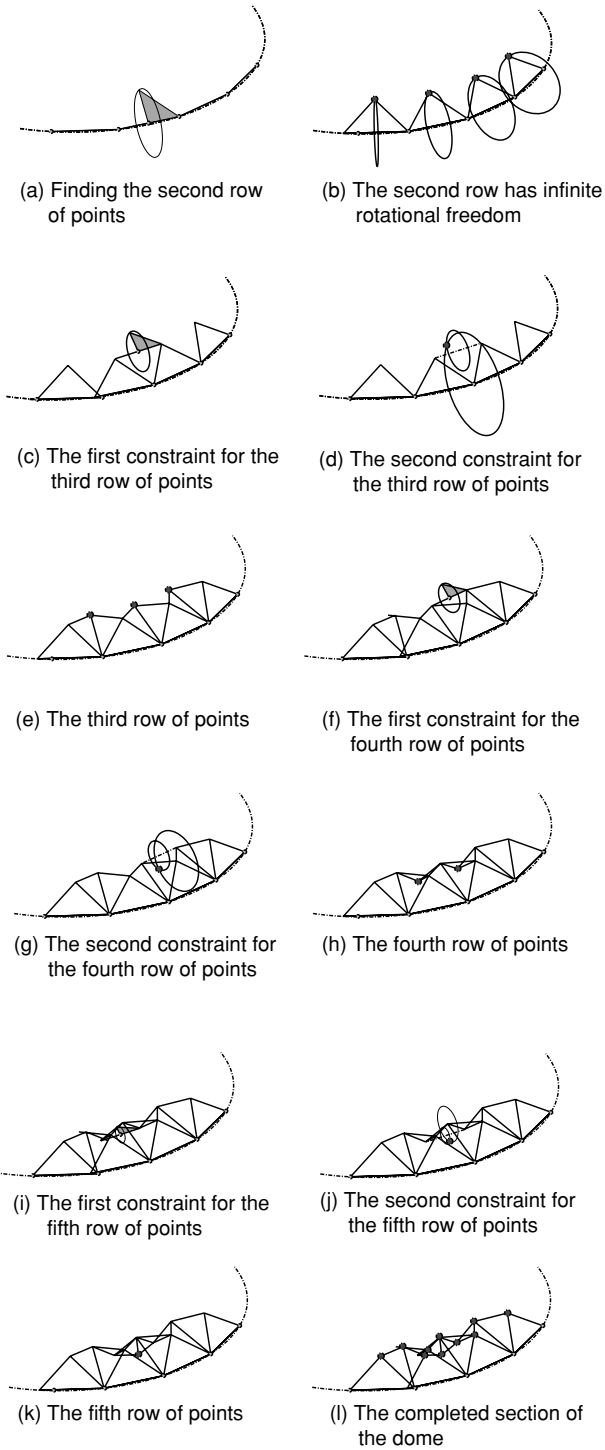


(b) The second constraint is that P_i must lie on a circle with radius equal to the length of the radial line between P_i and $P_i - 2$.



(c) The result of solving these two constraints is two points on the intersection of two circles.

We repeat this point-finding mechanism for each row of points, using the points on previous rows, except for rows one and two. (See Figure 15.)

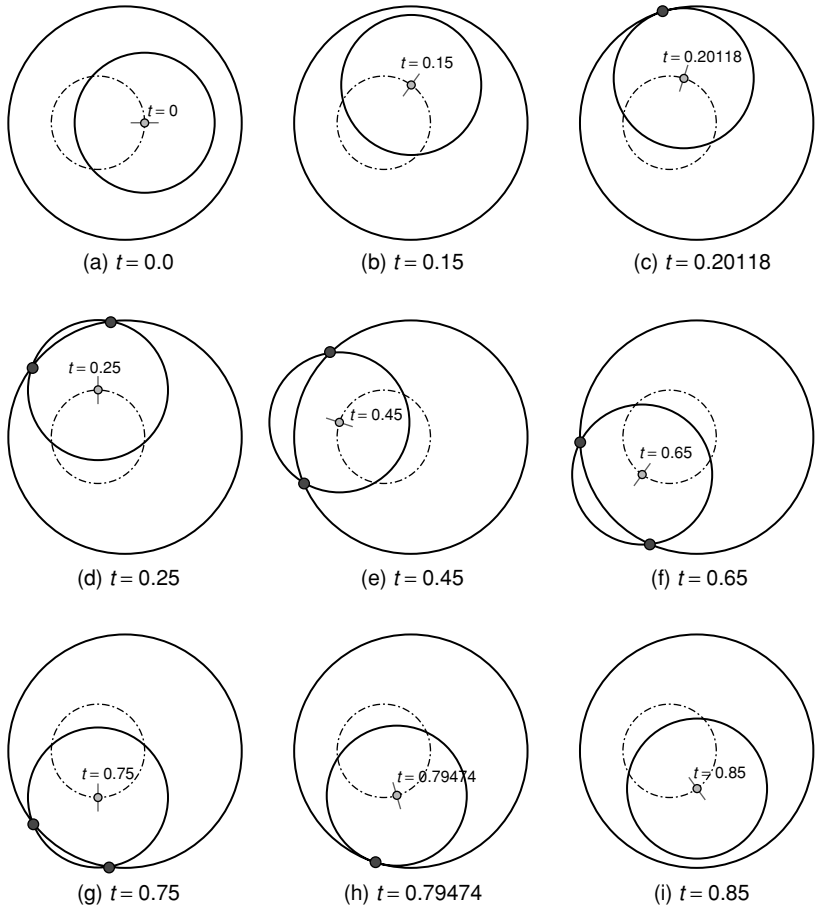


◀ Figure 15. The algorithm for creating representative points on all rows in a 18/5 dome.

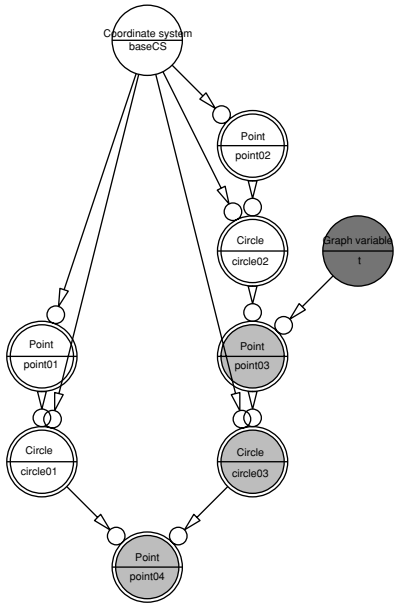
The first row of points comprises the points on the circle. The second row of points is different from the rows to come, in that it only has one of the two constraints, the same distance from the nearest two points on the first row. This lack of constraints, gives this row of points infinite rotational freedom, by which they can move on a circle. Adding each row of points limits this range more and more. If we put the second row points outside this range, the circles would not intersect and it would be geometrically impossible to find the points on the following rows. We need to find this range to compute a complete dome for each and possible value of *n* (number of points on the circle), *d* (jump), *R* (the 2D radius in the 2D pattern), and *r* (the final radius in the 3D model). Finding the feasible range is what requires a goal seeker.

Figure 16 shows a parametric model in which a parametric point on an inner circle placed asymmetrically within an outer circle locates a third circle. An intersection between the outer and the third circle only exists for some values of *t*, the parameter of the point on the inner circle.

► Figure 16. The feasible range of possible solutions. The circles intersect only when 0.20118 < T < 0.79474.



However, in the parametric model, *t* is upstream of the third circle. (see Figure 17) We must search the possible values of *t* (from 0 to 1) to find the range of *t* in which an intersection occurs.



◀ Figure 17. The symbolic graph of the 3 circle model shows that *t* is upstream of the third circle.

The goal seeker is used when "we don't know the input value that makes a specific output" [5] and the intent of a goal seeker is "to change an input until a chosen output meets a threshold" [5]. In the case of the dome, for each i^{th} row, a goal seeker finds the maximum and minimum positions on the circle for its points by varying the angle of rotation of the row.

The goal seeker script changes the position of the second row points on the circle, then updates the graph and checks the status of the points on the rows so far. If their creation is successful, the job is done. If not, the goal seeker slightly changes the position of second row points, and updates the graph and checks the points again. This loop continues to run until all the points are successfully created [5].

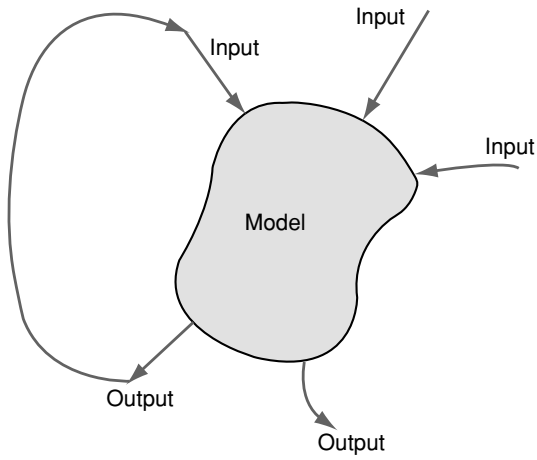
To do that, the goal seeker uses a combination of hill-climbing and binary search. While doing the search, the incremental change of the position may cause the second row points to pass the range. In that case, the script reverses and reduces the step change and continues the search again, until the final result is achieved. [5]

Any model in a propagation based system can be represented by a graph of nodes and arcs connecting the nodes. The nodes are objects containing dependent and/or independent variables and constraint between the variables. An arc that connects node 1 to node 2 means that a variable in node 2 is dependent on a variable in node 1. A node with no arc coming

into it is an independent node. Objects are typed and each type has update algorithms based on the dependent and independent variables of that object. To be computable with the standard propagation algorithms, graphs cannot have cycles since no topological sort of the graph is then possible and the propagation algorithm would thus not terminate [1].

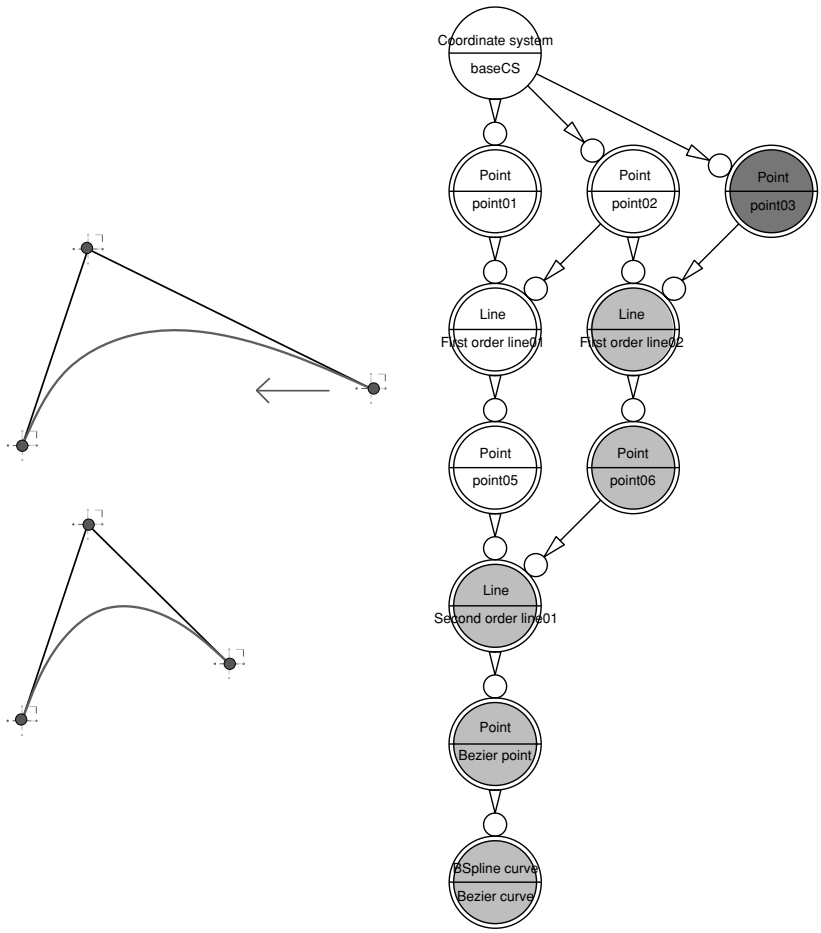
A goal seeker introduces a cycle by being able to let the status of a result control the value of an input.

► Figure 18. Goal seeker causes a cycle in the graph [5].



In propagation without goal-seeking, as soon as any independent variable in the graph changes, the graph is automatically updated, meaning that the nodes are visited and their values are computed using their update algorithm. In a simple example of an order three bezier curve, a change in one of the base points causes an update to the graph, so that all the objects dependent on that point in the model follow that change.

With a goal seeker, we take control of update graph by explicitly calling the graph updating function (UpdateGraph in GC). By doing this, we can specify when and where in the goal seeker script we want the graph to be updated and values to be computed.



◀ Figure 19: An order three bezier curve and its symbolic node graph in a propagation based CAD system.

(a) When an independent point moves, the graph is updated to reflect the change

(b) The independent point and its consequent nodes in the graph

Here is the algorithm for a general goal seeker written in pseudocode.

```
1    FUNCTION Side (result, target)
2    IF (result – target)/|(result – target) | > 0
3    RETURN TRUE
4    ELSE
5    RETURN FALSE
6    // TRUE if result > target AND FALSE if result < target
7    // The function Side will be particular to each specific goalseeker
8    FUNCTION Dist ( current Position, threshold)
9    RETURN the distance between the current state of the model and the desired
     state at the threshold
10   // This function determines how far the result is from the target
11   SET driver to the desired variable that is going to be changed
12   SET result to the variable or object property that you want to check
13   SET target to the threshold that the result should meet
```

```
14  INITIALIZE lastKnown
15  INITIALIZE lastKnownDist
16  SET incrementSubdivided to 0
17  // the number of times that incrementSubdevided is divided
18  SET incrementAdded to 0
19  // the number of times that the increment is added to the driver
20  SET increment to any reasonable value in the search context, such as 0.2
21  SET giveUpWhen to 100
22  SET closeEnough to 0.0001
23  SET currentSide = Side(result, target)
24  WHILE | increment | > closeEnough AND increment Subdivided < giveUpWhen
25  SET increment Added to 0 // initialization
26  WHILE Side(result ,target) = currentSide AND increment Added < giveUpWhen
27  // checking to see if we are still at the same side of the threshold that we were
    before and have not jumped over it
28  SET lastKnown to the driver
29  // always keep track of our last position before moving forward
30  SET lastKnownDist to Dist(result, target)
31  SET incrementAdded to incrementAdded +1
32  SET driver to driver + increment
33  Update the graph
34  IF | Dist(result ,target)| > | lastKnownDist | AND Side(result, target ) = currentSide
35  // if the result is getting farther from the target without passing it, then we need to
    change the direction of our search
36  increment = – increment
37  ENDIF
38  ENDWHILE
39  SET driver to lastKnown
40  // if we have jumped over the threshold , we go back to where we were before
    and start again with a smaller increment
41  Update the graph
42  SET incrementSubdivided to incrementSubdivided +1
43  DIVIDE increment by 2
44  ENDWHILE
```

In the context of the dome, the algorithm slightly changes to match the geometry. Since we are looking for a range, with a maximum and a minimum point, we need to run the goal seeker twice to find both ends of the range.

```
1   FUNCTION Side( point)
2   IF Success(point)
3   RETURN TRUE
4   ELSE
5   RETURN FALSE
6   // In a propagation–based system, each object is in either a " successfull" or an
    "unsuccessful" state depending on the las invocation of its update method.
7   FUNCTION Dist ( current Position ,threshold)
8   RETURN the distance between the current state of the model and the desired
    state at the threshold , in this case, the perpendicular distance between the two
    circles that should intersect to produce the Nth row point.
9   // This function determines how far the result is from the target
10  SET driver to the parameter (T) of the second row points of the dome
11  SET result to the Nth row of points in the dome (the last built row)
12  SET target to TRUE
13  INITIALIZE lastKnown
14  INITIALIZE lastKnownDist
15  SET incrementSubdivided to 0
```

```
16  SET incrementAdded to 0
17  SET increment to any reasonable value , such as 0.2
18  SET giveUpWhen to 100
19  SET closeEnough to 0.0001
20  WHILE | increment | > closeEnough and incrementSubdivided < giveUpWhen
21  SET incrementAdded to 0
22  WHILE Side(result) = target AND incrementAdded < giveUpWhen
23  // Checking to see if we are still inside the range and have not jumped over the
    threshold
24  SET lastKnown to the driver
25  // Always keep track of our last position before moving forward
26  SET lastKnownDist to Dist(result ,target)
27  SET incrementAdded to incrementAdded +1
28  SET driver to driver + increment
29  Update the graph
30  IF | Dist(result ,target)| > | lastKnownDist | AND Side(result)
    = target
31  // if the result is getting farther from the target without passing it, then we need to
    change the direction of our search
32  increment = – increment
33  ENDIF
34  ENDWHILE
35  SET driver to lastKnown
36  // If we have jumped over the threshold , we go back to where we were before and
    start again with a smaller increment
37  Update the graph
38  SET incrementSubdivided to incrementSubdivided +1
39  DIVIDE increment by 2
40  ENDWHILE
```

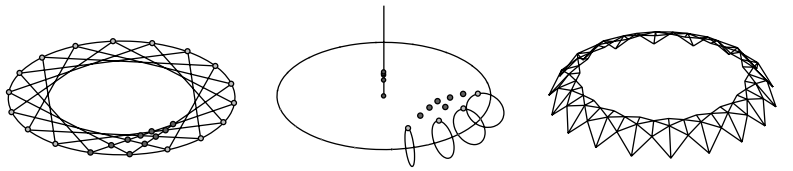
In order for the first condition to be satisfied and the loop to start, the second row points have to be at a position where the points in the current row are being successfully created. In other words, they have to be somewhere inside the range of feasible configurations. The following algorithm is another version of a goal seeker that puts the points inside the range. It needs to be run before the main goal seeker. This loop starts when the points are outside the feasible range. It moves the points, checks to see if it is moving towards the range and not in the wrong direction. If so, it keeps moving the point until it falls inside the range or it passes the range. If it passes over the range, it returns the point to the previous position and makes the step size smaller to avoid passing the range.

```
1   FUNCTION Side( point)
2   IF Success(point)
3   RETURN TRUE
4   ELSE
5   RETURN FALSE
6   // In a propagation—based system, each object is in either a “successfull” or an
    “unsuccessful” state depending on the last invocation of its update method.
7   FUNCTION Dist (current Position ,threshold)
8   RETURN the distance between the current state of the model and the desired
    state at the threshold , in this case, the perpendicular distance between the two
    circles that should intersect to produce the Nth row point
9   // This function determines how far the result is from the target
```

```
10  SET driver to the parameter (T) of the second row points of the dome
11  SET result to the Nth rwo of points in the dome (the last built row)
12  SET target to TRUE
13  INITIALIZE lastKnown
14  INITIALIZE lastKnownDist
15  SET incrementSubdivided to 0
16  SET incrementAdded to 0
17  SET increment to any reasonable value , such as 0.2
18  SET giveUpWhen to 100
19  SET closeEnough to 0.0001
20  INITIALIZE rangePassed
21  // rangePassed is true only when during the search , result passes the whole
    feasible range
22  WHILE | increment | > closeEnough AND incrementSubdivided < giveUpWhen
    AND result != target
23  SET incrementAdded to 0
24  SET rangePassed to false
25  WHILE result != target AND incrementAdded < giveUpWhen AND rangePassed =
    false
26  // Checking to see if we have passed the range in which case we have to go back
    and take a smaller step, or we are actually inside the range and we stop the
    search
27  SET lastKnown to the driver
28  // Always keep track of our last position before moving forward
29  SET lastKnownDist to Dist ( result , target )
30  // Keeping track of the last value of distance between result and target to be able
    to compare our new status with the previous one
31  SET incrementAdded to incrementAdded +1
32  SET driver to driver + increment
33  Update the graph
34  IF Dist(result ,target) * lastKnownDist < 0
35  // Which means we have passed over the range and need to go back and take a
    smaller step
36  SET driver to lastKnown
37  Update the graph
38  SET rangePassed to true
39  ELSE IF Dist(result ,target) > lastKnownDist
40  // Which means that we are moving in the wrong direction and we need to change
    direction
41  SET increment to —increment
42  ENDIF
43  ENDIF
44  ENDWHILE
45  SET incrementSubdivided to incrementSubdivided +1
46  DIVIDE increment by 2
47  ENDWHILE
```

There are three main steps in creating the dome in a propagation-base parametric system, such as Bentley's Generative Components, as shown in figure 20. First, we need to draw the 2D star pattern (all Rasmi domes originate from a drawing). From this pattern, we extract the lengths of the line segments, which will remain the same until the end. Second, we create the smallest section of the dome that includes the points in all rows. Considering that, in order to find a point in any row, we need the two

nearest points in the previous row, the smallest section in a dome of *n*/*d* starts with *d* points on the final circle and ends with one point on the d^{th} row. In this step we find the position of each row of points using *goal seeker*. We then use this data to create the complete dome in the third and final step by creating the points and connecting them.



(a) Step1: Drawing the 2D pattern

(b) Step 2: Finding the points in the smallest section of the dome, using the *goal seeker*

(c) Step 3: Creating the dome

◀ Figure 20. Three steps of creating the dome in Generative Components.

This algorithm can also be used in non-CAD systems, such as Processing and Microsoft Excel. In these type of software, we simply follow the same steps, repeating solving the geometrical equations inside a hill-climbing algorithm that we must write and storing the data, without really having to draw anything. A goal seeker is an inevitable part of the process in all of these systems. Once we have the algorithm, we can easily produce variations of the dome, by changing the different parameters as shown in figures 11 and 12.

Acknowledgements

This work was partially supported through the National Science and Engineering Research Council Discovery Grants Program, the BCcampus Online Program Development Fund and the Interdisciplinary Research in the Mathematical and Computational Sciences at Simon Fraser University.

References

1. Aish, R., and Woodbury, R. *Multi-level Interaction in Parametric Design*. Springer Berlin/Heidelberg, 2005, pp. 151–162.
2. El-Said, I. *Geometric concepts in Islamic art*. World of Islam Festival Pub. Co. Ltd, London, 1976.
3. Kaplan, C. S. Computer generated Islamic star patterns. In *the 2000 Bridges Conference* (Southwestern College, Kansas, 2000).
4. Lurzadah, H., Raiszadah, M., and Mufid, H. *Ehyaaye honarhaaye az yaad rafte: mabaani memaarie sonnati dar Iran*. Mawla, Tehran, Iran, 1995.
5. Woodbury, R. F. Design patterns for parametric modeling, 2007. accessed at www.designpatterns.ca on 26 June 2008.

Maryam M. Maleki and Robert F. Woodbury
School of Interactive Arts and Technology
Simon Fraser University Surrey
Central City
250 - 13450 102nd Avenue
Surrey, BC, V3T 0A3, CANADA
{mmaleki, rw}@sfu.ca