The 5th International Symposium on Virtual Reality, Archaeology and Cultural Heritage
VAST (2004)
K. Cain, Y. Chrysanthou, F. Niccolucci, N. Silberman (Editors)

Generative Parametric Design of Gothic Window Tracery

Sven Havemann Dieter W. Fellner

s.havemann | d.fellner@tu-bs.de
Computer Graphics Group, TU Braunschweig, Germany

Abstract

Gothic architecture, and especially window tracery, exhibits quite complex geometric shape configurations. But this complexity is achieved by combining only a few basic geometric patterns.

We present some principles of this long-standing domain, together with some delicate details, and show how the constructions of some prototypic Gothic windows can be formalized using our Generative Modeling Language (GML). The emphasis of our procedural approach is on modularization, so that complex configurations can be obtained from combining elementary constructions. Different combinations of specific parametric features can be grouped together, which leads to the concept of styles. They permit to differentiate between the basic shape and its appearance, i.e., in a particular ornamental decoration.

This leads to an extremely compact representation for a whole class of shapes, which can nevertheless be quickly evaluated to obtain a connected manifold mesh of a particular window instance. The resulting mesh may also contain free-form surface parts, represented as subdivision surfaces.

1. Introduction

Window tracery is the very particular type of window decoration found in any building of Gothic style. Gothic architecture, and especially window tracery, exhibits quite complex geometric shape configurations. But this complexity is achieved by combining only a few basic geometric patterns, namely circles and straight lines, using a limited set of operations, such as intersection, offsetting, and extrusions. The reason for this lies in the nature of the process how these objects have been physically realized, i.e., through constructions with compass and ruler. Consequently, Gothic architecture is a great, although challenging, domain for parametric modeling.

The traditional way to communicate the construction, e.g., of a particular window, is by a series of drawings. Over the centuries, these drawings have evolved into a very domain specific, condesed code, essentially a compressed communication form between architects and stone masons (Fig. 1). The construction process itself, however, has been based on extensive experience. It has never been formalized in an unambiguous way so that, e.g., a computer could reproduce results of equal quality.

The purpose of this paper is to (a) identify the basic operations used in window construction and decoration, and to (b) demonstrate how parameterized constructions can be concisely expressed in a formal language. Accordingly, the paper is structured in two parts. We consider these two steps as an important, and novel, paradigm for cultural reconstruction. To our knowledge, this has not been attempted before in an equally consequent manner, which makes it very hard for us to refer to prior work. This paper is the much extended version of a short paper that appeared on this year's Solid Modeling conference [HF04].

2. The construction of a Gothic Window

The basic pattern in Gothic Architecture is the *pointed arch*. Its geometric construction is based on the intersection of two circles. The circles are tangent continuous to the sides of an arch or a window, given as two vertical line segments (Fig. 2). Consequently the midpoints m_L (left segment) and m_R (right segment) of the circles lie on the horizontal line through the upper endpoints p_L and p_R of the left and right segments, the *arch basis points*. The pointed arch is symmet-

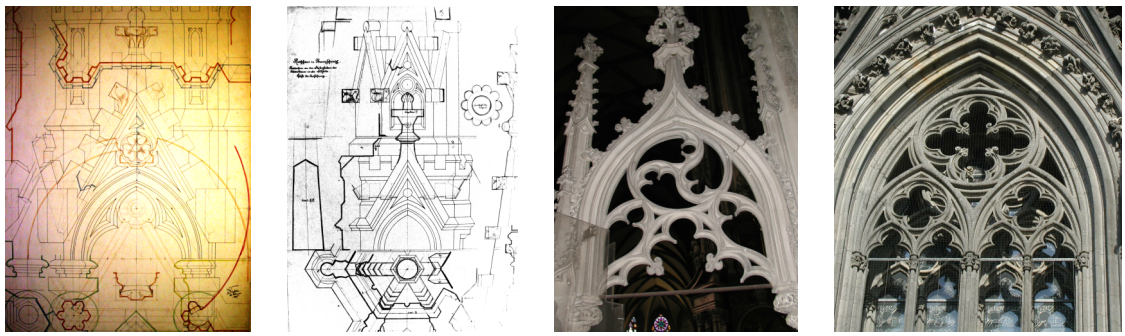

Figure 1: *For centuries, highly coded construction plans were turned into precise pieces of stone by skilled stone masons.*

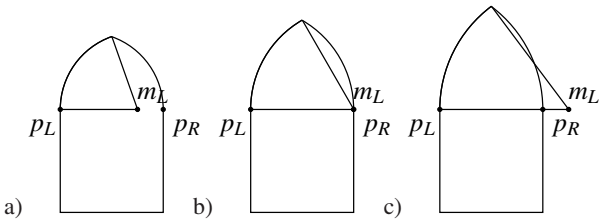

Figure 2: *Gothic arch with varying excess: Four-centered (0.75), equilateral (1.0), and pointed arch (1.25)*

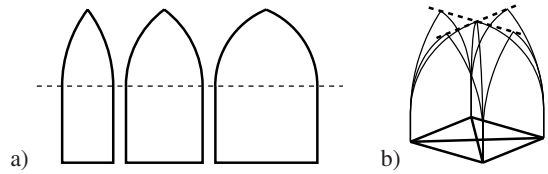

Figure 3: *The height of a pointed arch can be kept constant even when the width varies.*

ric, so both circles have the same radius $r = \mathrm{dist}(p_L, m_R) = \mathrm{dist}(p_R, m_L)$.

We call the ratio $r/dist(p_L, p_R)$ the *excess* of the arch. When the excess is 1.0, the circle midpoints coincide with the upper segment endpoints. Together with the circle intersection, they form an equilateral triangle. This is the standard pointed arch, also called *equilateral arch*. With an excess > 1.0, the circles intersect at a sharper angle, and this is what is actually called a *pointed arch*. When the excess is < 1.0, the arch is not so high, and this is called *four-centered arch*. The extreme case is the round arch with an excess of 0.5, so that m_L, m_R coincide in the midpoint between p_L, p_R.

2.1. Historical Development of Window Tracery

The pointed arch is a generalization of its predecessor, the *round arch*. It was a technological breakthrough that, after its introduction around 1140, has truly revolutionized the construction of cathedrals. It was first systematically employed by abbot Suger in the cathedral of St. Denis (near Paris, France), and the new style spread over all Europe in just a few decades. It has dominated the European sacral architecture for more than two hundred years, and gave rise to a veritable footrace between cities, with cathedrals becoming ever more sophisticated and risky.

Technologically, the great advantage of the pointed arch over the round arch lies in the fact that the distance between the columns could now be varied without affecting the height of the arch (Fig. 3 a). This leaves greater flexibility for positioning the columns, and helps to solve delicate problems with the design of the ground layout in a cathedral (Fig. 3 b).

Basically the same shape as for an arch can also be used for a window. The idea of Gothic cathedrals is to make the walls of the church as transparent as possible, in order to let a maximum of light enter the room. With coloured windows, a cathedral was flooded with light in all colors, which was one of the manifestations of God in the perception of the medieval christian. The size of the windows in relation to the size of the walls increased, and the walls actually "dissolved" to the point where they completely lost their supporting function. Gothic cathedrals get their stability almost exclusively from columns, and not from walls [Bin02].

There is a remarkable development of the ornamental decoration in the upper part of the window, the *couronnement* (Fig. 4). In the Early Gothic period, starting around 1140, the windows were created by cutting openings into large stone plates in the wall. This premature form of window tracery is therefore called *plate tracery*. In the High Gothic period, from around 1250, the stone parts became ever thinner, and the windows covered an increasing portion of the wall. The glass windows were set into a network of individual stones, the *bar tracery* (Fig. 4 c,d). The late Gothic period, in the 14th and 15th centuries, saw a great refinement and sophistication of window tracery. The basic patterns were varied over and over again, with recursive sub-structures and self-similarity, to the point where the static stone appeared to be

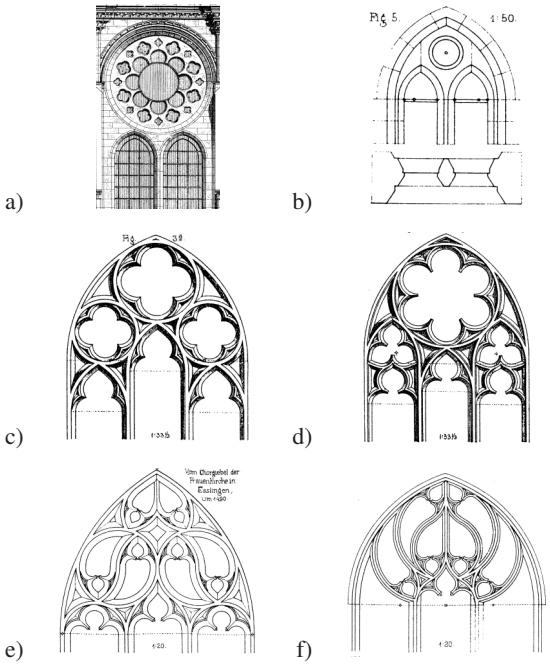

Figure 4: *Evolution of Gothic window tracery. Early period (a,b), High Gothic (c,d), late period (e,f).*

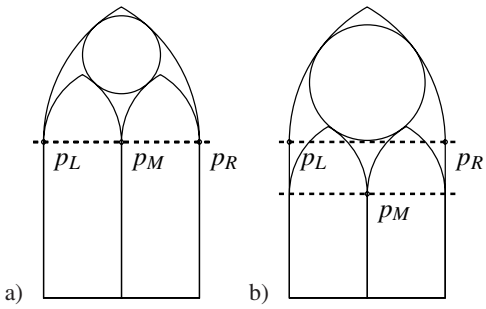

Figure 5: *Geometry of the prototype window*

actually flowing. An example is the French and English *flamboyant* style (Fig. 4 e,f) with its flame symbolics [Bin89].

2.2. The Prototype Window

As our prototype window, we chose a very common and basic High Gothic window type, one with two sub-windows that are also pointed arches (Fig. 5 a). It exhibits the main shape features, the "shape vocabulary", that was subsequently refined and varied in the Late Gothic period. The sub-windows have most often the same excess as the large arch, which makes the excess a high-level parameter of the window.

The window shown in 5 (a) has excess 1, so that the midpoints of the circular arcs and the basis points p_L, p_R co-

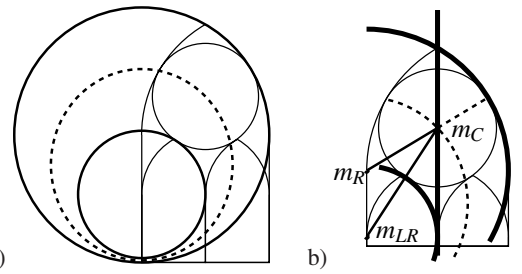

Figure 6: *Determining the midpoint and radius of the rosette's circle.*

incide. They are also basis points of the subwindow arches. But it is often the case that the sub-arches are set down with respect to the big arch, in order to create a bigger space for the couronnement (Fig. 5 b). So the vertical distance between the basis points of the outer and inner arches is another high-level parameter of the window.

2.3. Adding a Circular Rosette

The space between the outer and inner pointed arches can be filled in many different ways. This decoration is the distinguishing feature of each individual window. In early days of Gothic, this space was quite often filled with a circular rosette. Geometrically, the problem is to find midpoint m_C and radius r_C of a circle that touches the outer arch and two segments of the inner arches. Consider the set of all points in between a big arc and a sub-arc, for instance arc_{LR} and arc_R, as in Fig. 6 a). The points that have the same distance to both respective circles, (m_R, r_R) and (m_{LR}, r_{LR}), are shown as dotted curve.

This curve is an ellipse, which is revealed in Fig. 6 b): Connect a point on it, for instance m_C, with both midpoints m_R and m_{LR}. The distance from m_C to the upper midpoint m_R is less than the radius r_R of the big circle (dotted continuation), so $\mathrm{dist}(m_C, m_R) = r_R - x$ for some $x > 0$. Similarly, the distance from m_C to the lower midpoint is $\mathrm{dist}(m_C, m_{LR}) = r_{LR} + y$ for some $y > 0$. But m_C has the *same* distance to both circles, so $x = y$. Then x cancels out from the sum of both distances, and $\mathrm{dist}(m_C, m_R) + \mathrm{dist}(m_C, m_{LR}) = r_R - x + r_{LR} + x = r_R + r_{LR}$ is constant. Since this holds for all points on the dotted curve, it must be an ellipse, and m_R and m_{LR} its foci. The midpoint m_C of the rosette can be obtained as an intersection between this ellipse $(m_R, m_{LR}, r_R + r_{LR})$ and the vertical axis of symmetry. This is practically computed by intersecting a unit circle with an affinely transformed line.

2.4. Offset Curves

The rosette circle and the sub-arches partition the window into disjoint regions. These regions define the basic structure of the window, which is then further refined. This can

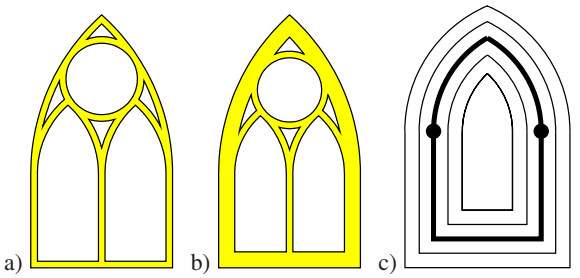

Figure 7: *a),b) The regions are shrunk to embed them in a common border plane. c) The offset operation changes the excess of a pointed arch, but keeps the circle midpoints constant.*

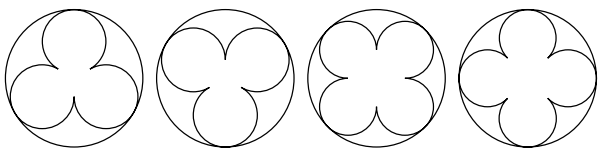

Figure 8: *Lying and standing trefoil and quatrefoil rosettes.*

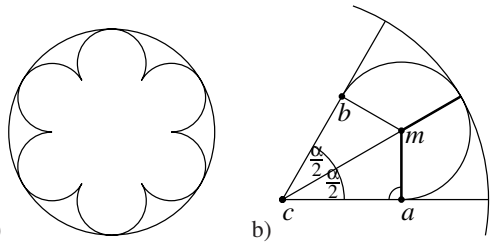

Figure 9: *Construction of a rosette with six rounded foils, so* $\alpha = \frac{2\pi}{6}$.

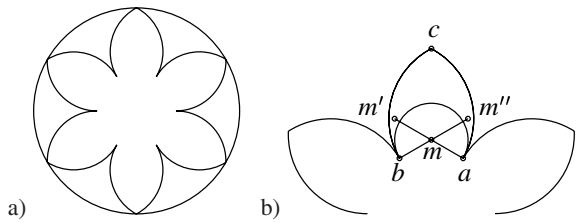

Figure 10: *Construction of a rosette with pointed foils, with a relative displacement of 1.15 to obtain* m' *and* m'' *from m.*

be done by simply adding a profile around the actual window holes to emphasize the shape, or, especially in the later period, by adding sub-structures, again composed of lines and circular arcs. It is very common that there is a thin planar border between adjacent regions, so that there is actually a connected border plane. Geometrically, this means that the region boundaries are offset by a certain distance, as depicted in Fig. 7 a),b), so that a contiguous border plane results, shown in yellow.

Regarding the great variety of examples where this pattern is used, it is reasonable to distinguish between two different offset parameters, namely the interior offset and the offset distance of the ensemble to the outer pointed arch: Both parameters are equal in Fig. 7 a), while in 7 b) the outer offset is doubled.

The great thing about the circle is that it is so easy to compute its offset. This applies also to curves that are created from a sequence of several circular arcs and line segments. But note that if the sequence contains corners, e.g. two arcs joining in an intersection of the respective circles, the intersection of the offset circles must be computed for obtaining the offset curve sequence. Simple scaling is not sufficient to create offset curves: The offset of a pointed arch has a different excess than the original arch, as shown in Fig. 7 c).

2.5. Rosette Window with Multiple Foils

A very common way to fill a circular region is by an rosette with multiple foils, for example a trefoil or a quatrefoil. We consider two variants of rosettes, namely with round and with pointed foils. A further distinction is between *lying* and *standing* rosettes, as shown in Fig. 8.

The geometry is fairly straightforward: Given the number n of foils in a unit circle, the radius r of the round foils is computed as shown in Fig. 9 b). Consider the tangent from center c to the circle (m, r). The distance from c to m is $1-r$, so the length of the perpendicular from m to the tangent is $(1-r)\sin\frac{\alpha}{2}$, which is supposed to be r. This equation gives $r = \sin\frac{\alpha}{2}/(1+\sin\frac{\alpha}{2})$. The perpendicular feet a and b are the endpoints of the circular arc that is rotated and copied n times to make up the rosette.

The pointed foils can be obtained from the round foils as shown in Fig. 10. The midpoints m', m'' are obtained from m by displacement along the lines (a, m) and (b, m). The pointedness, and thus the radius of the circles, is influenced by the amount of displacement, which can be specified in relation to the original radius. Points a and b and the intersection point c then specify the arcs which make up the pointed foils.

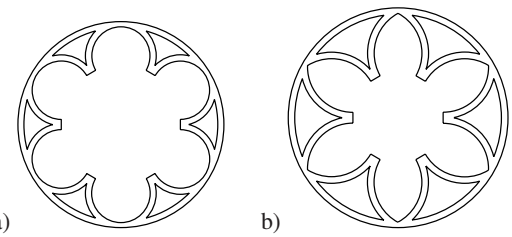

Figure 11: *Offset operation to obtain the region borders.*

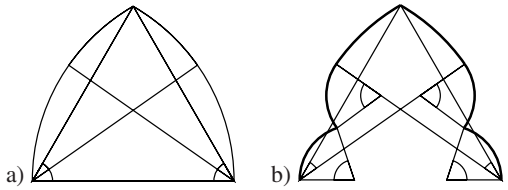

Figure 12: *Pointed trefoil arch obtained from pointed arch.*

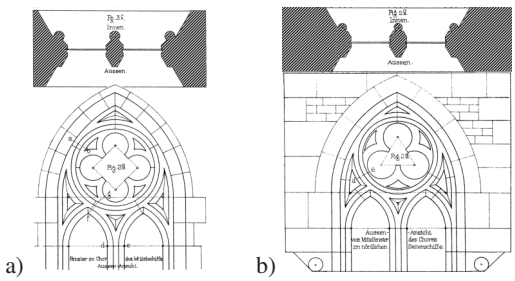

Figure 13: *Windows with horizontal profile cut (above).*

In order to fit into the original circle, the foils are scaled: Circles permit uniform scaling, and so do circular arcs.

Just as described in section 2.4, a connected boundary region is constructed from the network of circular arcs. Examples are shown in Fig. 11. Note that also these offset curves also only consist of arcs and line segments.

2.6. Further Refinements

Circular arcs can be combined very flexibly. Both corners and tangent continuous joints can be obtained from quite elementary geometric constructions. The pointed trefoil arch in Fig. 12 for instance is easily obtained from a pointed arch: First both arcs are symmetrically split, and then the lower parts are replaced each by a pair of smaller arcs. Tangent continuity is obtained by choosing the midpoint of the next arc on the line through mid- to endpoint of the given arc. This example shows the source of the great variability of geometric patterns in Gothic architecture.

2.7. Appearance: Profiles

So far, the structure of the window is solely defined by a few two-dimensional regions whose boundaries are composed of circular arcs and line segments. The fascinating and impressive three-dimensionality of Gothic windows is achieved by *profiles* that give depth to the two-dimensional geometric figures. In architectural illustrations, profiles can often be found above or below a front view, like those in Fig. 13 from Egle [vEF96]. Note that the profiles are also composed of line and circle segments.

Technically, these profiles are swept along the region border curves, the profile plane being orthogonal to the tangent of the curve. At corner points, where the tangent is discontinuous, the sweep is basically continued onto the bisector plane. This is the plane that is spanned by the bisecting line of the angle in the corner and the normal of the 2D construction plane. Yet this is only the case when the curve is locally symmetric to the bisector plane. In more general cases, the discontinuity locally follows the *medial axis* of the two parts of the curve.

3. Re-Construction of Gothic Windows

The previous section gives an example for an analysis of a rich class of complex three-dimensional shapes, in order to understand how they were composed in principle. Cultural reconstruction on the other hand aims at (re-)synthesizing particular shapes based on assumptions derived from findings and artifacts. To let a computer create those hypothetic shapes, the respective class of shapes must be formalized in an un-ambiguous way. This raises the question for a suitable formalism. Our approach uses the GML.

The Generative Modeling Language (GML) is a simple, stack-based, interpreted programming language for creating polygonal meshes [Hav03]. The language core is very similar to that of Adobe's PostScript. This core is concisely described in the PostScript "Redbook" [Ado99], especially in chapter 3, which to a large extent also applies to the GML.

The GML does not have PostScript's extensive set of operators for typesetting, though. Instead, it provides quite a number of operators for three-dimensional modeling, from low-level Euler operators for mesh editing, to higher-level modeling tools, such as different forms of extrusions, and operators to convert back and forth between polygons and mesh faces.

Despite its simplicity, the GML is a full programming language, and it can be used to efficiently exploit the striking similarity between 3D modeling and programming: Both well-structured 3D models and good computer programs are composed of re-usable *modules*. Yet of course, archeologists are not urged to become programmers now: Methods exist to almost completely hide the GML code from the end users. Instead, the question in focus is whether the GML is capable of representing object descriptions and shape classes efficiently.

3.1. The Pointed Arch

A circular arc can be specified by start- and endpoint, and the midpoint of the circle, like in Fig. 9 b). In the GML, this is an array of three points, written `[ a m b ]`. To distinguish between the two possible arcs from a to b, we specify a normal vector n with the convention that the arc is always CCW oriented when the normal points to the viewer. Now given two circles (m_L, r) and (m_R, r) as in Fig. 2, the circle intersection is computed by an operator that expects midpoints and

radii and the plane normal on the stack: Every GML operator pops its arguments from the stack, computes one or more results, and pushes these back on the stack. In this case, the intersection points above and below the line between m_0 and m_1 are pushed on the stack, in this order. As we are only interested in one of them, the other one is popped. The two arcs are assembled with just one more line of code:

```
mL r mR r nrml intersect_circles pop !q
[ :q mL pL ] [ pR mL :q ]
```

The intersection is stored in a named register `q`. One drawback of stack-based programming is that it can be very tedious to keep track of the stack items and their order. Alternatively, variables can be stored in dictionaries – but searching through the dictionary stack may be slow. As a third alternative, the GML provides *named registers* for local variables: Values can be set using `!q` and retrieved using `:q`. This encourages a more "procedural" programming style, while still retaining the flexibility of the stack for parameter passing. A sequence of such operations can be merged into a single function, which then behaves like a built-in operator:

```
{ !nrml !offset !excess !pR !pL
  :pR :pL :excess line_2pt !mR
  :pL :pR :excess line_2pt !mL
  :pR :mR dist :offset sub !r
  :mL :r :mR :r :nrml intersect_circles pop !q
  [ :q :mL   :pL :pR :offset move_2pt ]
  [ :pR :pL :offset move_2pt   :mR :q ]
} /compute-arcs exch def
```

The first line of code just retrieves some parameters from the stack (in reverse order). The second line computes the midpoint m_R of the right arch as $p_R + excess \cdot (p_L - p_R)$. The radius r of both circles is computed as the distance from p_R to m_R minus the desired offset. The circular arcs are assembled "inline": An open bracket `[` is just a literal symbol that is put on the stack. The `move_2pt` operator moves a point into the direction of another point by a specified amount of units. The closed `]` just makes the interpreter create an array from stack elements until it finds the `[`. The family of pointed arches in Fig. 7 b) was created using this function by varying the offset parameter.

3.2. Creating a Closed Polygon

The resulting two arcs can be seen as a high-level representation of a pointed arch. In order to create a polygonal mesh, the arcs are converted to polygons: The `circleseg` operator expects an arc, a normal vector, an integer resolution, and a mode flag on the stack, and pushes a polygon. The advantage of the stack is the possibility of *operator chaining*: Calling `compute-arcs` creates two arcs on the stack, so only the other required parameters must be provided. To also polygonize the other arc, the topmost stack elements are swapped using `exch`, and the two point arrays, or polygons,

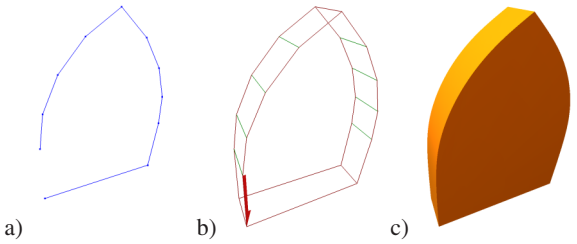

Figure 14: *Creation of a pointed arch: a) The circular arcs are sampled and combined into a single outline polygon. b) A double-sided Combined BRep face is created and then extruded. Sharp edges are shown in red, smooth edges in green. c) Front and back are* sharp faces *as their border contains BSpline curve segments. Quadrangles on the sides are tesselated using Catmull/Clark subdivision.*

are concatenated into a single polygon using the `arrayappend` operator:

```
nrml 4 1 circleseg exch
nrml 4 1 circleseg arrayappend
[ bL dup bR dup ] arrayappend
```

The order is always very crucial when operating on the stack: The arcs were pushed in order left, right, and they are processed right, left, to create a combined CCW oriented polygon. Note that it contains the tip of the arch twice, as the last point of the right arc and the first point of the left arc. This was done intenionally to create a corner there (see below). The bottom corners b_L, b_R of the segments are also duplicated and appended, so that the result is just a single polygon (Fig. 14 a).

Functions and executable arrays are synonymous in the GML: The opening brace `{` puts the interpreter in deferred mode, and subsequent tokens are not executed but just put on the stack. The closing `}` is a signal to create an ordinary array from these tokens, and to set its executable flag to true. So this is an executable array, ready to be executed as the loop body.

3.3. Creating a Polygonal Mesh

Two very basic modeling operations turn this polygon into a mesh which is shown in Figs. 14 b) and c):

```
5 poly2doubleface (0.0,1.0,5) extrude
```

The `poly2doubleface` creates a double-sided face from a point array, i.e., from a polygon. It provides different modes to do this. So in addition to the polygon it expects an integer number on the stack that specifies how to handle successive identical points, and whether the face should have *smooth* or *sharp* edges. Mode 5 creates sharp edges and crease vertices by default, but if a point occurs repeatedly, only a single *corner vertex* is created from it. The resulting double-sided face

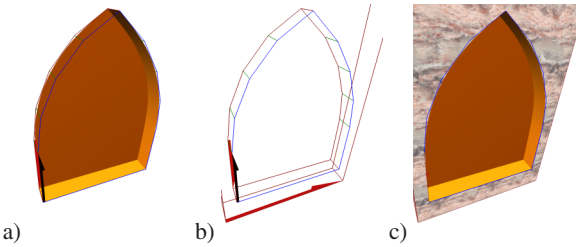

Figure 15: *Creating a window in a wall: a) The arch is inverted by negative extrusion. The backface (culled, not rendered) is closer and CW oriented (black halfedge). b) A wall with its front face in the same geometric plane as the arch's backface. The backface (black halfedge) is made a ring of the wall's front face (halfedge at bottom) using the* `killFmakeRH` *Euler operator. c) The tesselation of the wall respects the ring just created and trims it out.*

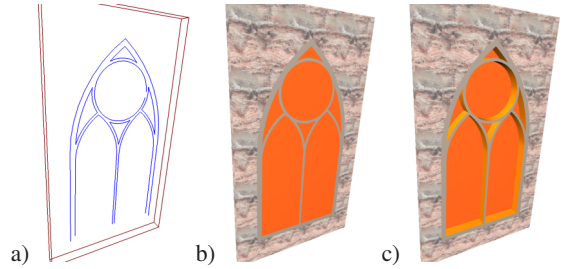

Figure 16: *a) The execution of* `gothic-window` *yields eight polygons on the stack defining the structure of the prototype window. b) The seven interior polygons are embedded in the face created from the outer arch. c) Simple negative etrusion of the interior regions.*

(with front and back) is topologically a solid, but it has zero volume. One mesh halfedge, as a handle to the face, is left on the stack. Halfedges are an integral data type in the GML.

Besides such an halfedge, the `extrude` operation expects a 3D point $(dx, dy, mode)$ on the stack, where dx specifies the shrinking and dy the vertical distance of the extruded face to the original face. The *mode* flag again specifies vertex types and edge sharpness flags. Mode 5 for example means that for "vertical" edges (along normal direction) the vertex type determines the sharpness, and the "horizontal" edges (in the displaced face plane) are sharp. This is why just the vertical edges from the three corner vertices are sharp in Fig. 14 b). This image also shows the halfedge returned by `extrude` as the red half-arrow in the lower left.

3.4. Mesh Representation: Combined BReps

The underlying mesh representation is the *Combined BRep* [HFon], *CBRep* for short, which was first presented on VAST 2001 [HF02]. Each edge in a CBRep carries a sharpness flag, that can either be set to *sharp* (red) or *smooth* (green). These flags determine which tesselation method to use for a face: Faces that contain one or more smooth edges are *smooth faces*. They are treated as Catmull/Clark subdivision surfaces. Faces with only sharp edges and where all vertices are corners are treated as *polygonal faces*. Faces with only sharp edges but where not all vertices are corners are *sharp faces*. The border of such a sharp face contains BSpline curves, but it is triangulated for display just like polygonal faces. An example is the front face of the arch in Fig. 14 c). A vertex is classified as *corner vertex* if three or more of its edges are sharp, with exactly two sharp edges it is a *crease vertex*. So the classification actually proceeds from edges to vertices to faces.

Combined BReps have another feature, which is that polygonal or sharp faces may have *rings*. A ring is a "negative", CW oriented, face in the interior of another face, which is the ring's *baseface*. It trims out a hole, which is quite useful e.g. for creating windows in a façade.

Note that the backface of the arch model is actually CW oriented from the perspective in Fig. 14 c). So one way to create a hole is by *negative extrusion*, so that the model appears inverted when backface culling is active (Fig. 15). A halfedge of the backface is obtained using "halfedge navigation": The `mate` operator returns the mate of a halfedge, which is part of the neighbour face, and runs in reverse direction. Given an edge `eWall` of the wall, the backside is turned into a ring of the wall using the `killFmakeRH` Euler operator, which reads "kill Face, make Ring Hole":

```
5 poly2doubleface dup mate exch
(0,-0.3,5) extrude exch eWall killFmakeRH
```

Here, the halfedge on the front-facing side of the double-sided face is duplicated, and the duplicate is flipped on the backside. The top two stack elements are exchanged, so that the negative extrusion of -0.3 units operates on the front face and pushes it farther away, returning a halfedge of the extruded face. Another `exch` brings the backside-halfedge again to the top, and it is made a ring of the wall's front side.

3.5. The Prototype Window in GML

The GML representation of the prototype Gothic window is of course a function `gothic-window` that expects the essential parameters as described in section 2.2 on the stack: The left and right base points of the arch, the height of the vertical segments of the lower part of the window, the excess of the arch, the plane normal, the vertical offset of the sub-arches (cf. Fig. 5), and the outer and inner border thicknesses. The following call produces the eight polygons that are shown in Fig. 16 a) together with the wall from before:

```
(-1,0,2) (1,0,2) 4.0 1.25 (0,-1,0)
0.2 0.1 0.05 gothic-window
```

Note that the cryptic sequence of parameters basically corresponds to what one would enter as values in a dialogue box in an interactive program. For the user not to mistype, the parameters in a dialogue box are named. But in background, this might very well create a line of code as the one above, in order to trigger the creation of geometry, when the user pushes the OK button.

The `gothic-window` function uses only operators and techniques that were presented so far. The only additional operator needed is `intersect_line_ellipse`, to compute the midpoint of the circle in the top from two points specifying the line, the two focal points of the ellipse, and its radius. The computation of the four fillets, each composed of three arcs, is done explicitly: The start-, mid-, and endpoints of each arc are uniquely defined by the intersection of circles. Those are derived from the big arch, the sub-arches, and the circle, based on the inner and outer border thicknesses.

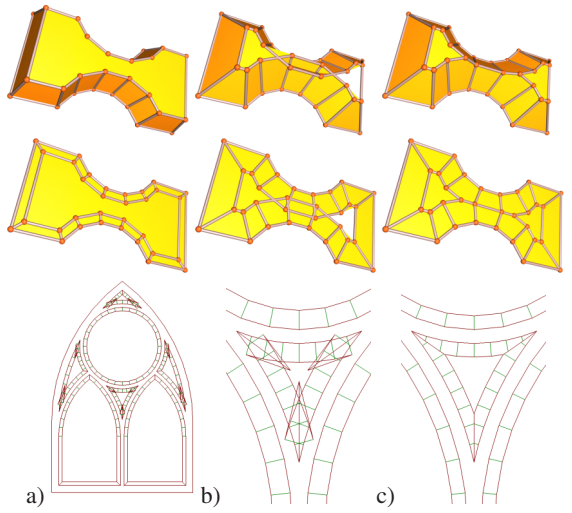

Figure 17: *a) Profiles are created by also shrinking an extruded face. b) A simple horizontal translation of the edges can yield self-intersections of the surface c) The more elaborate* `extrudestable` *operator removes self-intersections.*

3.6. Window Styles

Executing `gothic-window` creates eight polygons on the stack, in a defined order. They can be used to directly create mesh faces, in a similar manner as in section 3.3. But much more flexibility is gained by using *window styles*. Note that the eight polygons fall into four groups: Main arch, sub-arches, circle, and fillets. Instead of using `gothic-window` directly, one can wrap this into another function `gothic-window-style` that subsequently calls four functions corresponding to the four groups:

```
{ !eBack !eWall
  gothic-window
  :eWall             style-arch !eArch
  :eArch :eBack      style-circle
  2 { :eArch :eBack style-sub-arch } repeat
  4 { :eArch :eBack style-fillet   } repeat
} /gothic-window-styles exch def
```

This function expects the same parameters as `gothic-window`, and additionally two halfedges of the front- and backside of the wall where the window is to be inserted. After the polygons are created, a function `style-arch` is expected to take the topmost main arch-polygon from the stack, to somehow insert it into the wall, and to return a halfedge to the new face. The other items, the sub-arches, the circle and the fillets, are then inserted into this main arch face. Each of these functions is supposed to take two halfedges and one polygon from the stack, so that finally all polygons are popped and processed.

One great thing GML has inherited from Postscript is its quite flexible name lookup mechanism – namely the *dictionary stack* technique. When the GML interpreter encounters an executable name, such as `style-arch`, it searches top to bottom through the current dictionary stack to find the first dictionary that contains this name as a key. If it exists,

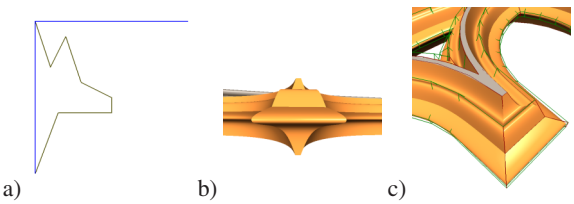

Figure 18: *a) Profile used for multiple extrusions. Extrusion starts in the origin (top left corner); x-direction defines shrinking offset, y-direction pushes into the wall. b) Shape created by sweeping the profile along the border of a ring. c) View more from above.*

the interpreter takes the corresponding value (literal or executable), and executes it. The dictionary stack can be freely manipulated at any time, using `begin` and `end`.

So if, e.g., a dictionary `My-Simple-Style` is active where all four `style-arch` keys are defined, pointing to suitable functions, a particular window instance can be created just like this:

```
My-Simple-Style begin
(-1,0,2) (1,0,2) 4.0 1.25 (0,-1,0)
0.2 0.1 0.05 gothic-window-style
end
```

3.7. Adding Profiles

The simple `extrude` operator used so far can also be used to shrink the extruded face. This is done by a simple horizontal offset, translating each edge to the left (i.e. towards the face

interior) by a specified amount of units. This works well in most cases (Fig. 17 a), but it can lead to self-intersections in faces with sharp corners, such as the central fillet (17 b). We have therefore added a more sophisticated `extrudestable` operator, based on the *straight line skeleton*, which is related to the *medial axis* of a polygon [EE99]. This operator not only removes self-intersections, it can also handle multiple extrude-operations at a time, i.e., an array of $(dx, dy, mode)$ triplets. This essentially defines a profile, see Fig. 18 for an example.

4. Results

We have carried out a number of experiments to test the variability of our style library, some of which are shown in Fig. 20. In 20 a) only the basic structure is used, and the same profile is used for fillets, circles and sub-arches. The openings are created by negative extrusion until the backside of the wall is reached, and then the extruded face is made a ring of the backside. In 20 b), the circle style is set to contain a rounded rosette, but with the same profile as before. In the next model 20 c), we switch to pointed foils in the rosette and to a more elaborate profile (the one from Fig. 18). The model 20 d) has just the same structure as before. Only the profile is different, the number of foils is set to 4, and the sub-arches use a pointed trefoil (cf. Fig. 12). In 20 e) only the profile for the sub-arches is varied, and another set of parameters is used for the pointed trefoil.

The next example, Fig. 20 f), is actually quite interesting: This time we use the fillets and the rosette from 20 c), and combine them with the style for the sub-arches from 20 d). So we can make use of the recursive structure of Gothic architecture, where the sub-arches are pointed arches just like the outer arch, and consequently permit the same type of refinement. The next two images show one further iteration: First a style with 20 c) in both sub-arches (20 g), and then this new style is again used for the sub-arches (20 h).

At the second refinement level of this model, the `extrudestable` operator has to remove quite a bit of self-intersections that would otherwise destroy the model, as can be seen in 21 a) showing a 3rd level rosette. This is due to the fact that in this style the profile, which is essentially a 2D polygon, is uniformly scaled by an amount depending on the wall thickness. Fig. 21 b) shows the tesselation and gives an idea of the number of triangles that are created by the subdivision surfaces (about 7 million, after 4 subdivision steps). – The final example 21 d) marks one area of future work: Given an image of a window like in 21 c), how well can we actually reproduce the existing shape? And could there be ways to determine some of the shape parameters automatically?

It should be mentioned that the complete GML code for all examples, styles, and the library of basic Gothic window tools such as `pointed-arch`, `rosette`, etc. fits into an ascii file of 27KB. Building the most complicated example window, 20 h), takes not much more than a second or two on a state-of-the-art PC. – An interactive GML demo can be downloaded from the GML website [Hav03] for verification – and for enjoying Gothic window tracery, of course.

Figure 19: *Varying excess with constant height. With a style library, a manifold of windows is obtained using just a few high-level parameters. The mesh is robust against invalid parameters (last image), the parameter validity should be checked on a higher software layer.*

References

[Ado99] ADOBE SYSTEMS INC.: *PostScript Language Reference Manual*, 3rd ed. Addison-Wesley, 1999. 5

[Bin89] BINDING: *Masswerk*. Wiss. Buchgesellschaft, Darmstadt, Germany, 1989. 3

[Bin02] BINDING: *Hochgotik*. Taschen Verlag, Cologne, Germany, 2002. 2

[EE99] EPPSTEIN, ERICKSON: Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete & Computational Geometry 22*, 4 (1999), 569–592. 9

[Hav03] HAVEMANN: The gml homepage, Aug. 2003. http://www.generative-modeling.org. 5, 9

[HF02] HAVEMANN S., FELLNER D. W.: A versatile 3D model representation for cultural reconstruction. In *Proc. VAST 2001 Intl. Symp.* (Glyfada, Greece, 2002), ACM Siggraph, pp. 213–221. 7

[HF04] HAVEMANN S., FELLNER D.: Generative parametric design of gothic window tracery. In *Proc. Shape Modeling and Application (SMI'04)* (Genova, June 2004), Giannini F., Pasko A., (Eds.), IEEE, pp. 350–354. 1

[HFon] HAVEMANN, FELLNER: Progressive combined breps – multi-resolution meshes for incremental real-time shape manipulation. *Computer Graphics Forum* (submitted for publication). 7

[vEF96] VON EGLE, FIECHTER: *Gotische Baukunst*, reprint from 1905 ed., vol. Bd. 3 of *Baustil- und Bauformenlehre*. Verlag Th. Schäfer, Hannover, Germany, 1996. 5