A Comprehensive BIM-based Parametric Model


The physical models in Figure 1 are outcomes of a design project in the linked studio (led by Prof. Rodney Hill at Texas A&M University). The algorithm designed by the student and applied to the design of both the center and outside structures is: at every new level, the element with a given setback value is rotated so that its corner points will touch exactly the corresponding edges of the element at the previous level.

This is a comprehensive example demonstrating how to create a BIM-based parametric model for resolving a real-world design problem. In this example, we can exercise: Revit massing, family, shared parameters, formulas, trigonometry (tangent arc-tangent), algebra (polynomial/quadratic equations), Access MDB exporting/importing, VBA to update the MDB. Currently Revit doesn’t provide a way to store the family data like the accumulated sum of angles of all current and previous units. That is why the MDB is used for doing this calculation and updating the model.

This model was a good example to demonstrate the parametric modeling and therefore built the digital model (though normally the order should be flipped if using CNC fabrication). The parametric models show that (1) how mathematics is used in the design. Quadratic formula solutions and trigonometry functions are used to resolve the mathematical problem for calculating the rotation angle for each level (the angle increment is different at each level); (2) it fosters design changes to be made easily; (3) the apparent differences between the computer models are the results of different parameters following the same algorithm – this helps student comprehend the simple essence of the seemingly complex forms; (4) the precision of the computer model will enable the physical model to be precisely created with CNC machines; and (5) physical materials can be applied to the geometry (e.g. in Figure 2 upper-right), indicating that the structure can potentially be built, which is a significant advantage of BIM-based parametric modeling, over other geometry-based parametric modeling methods.

Figure 1
Physical models of a lighting appliance designed and created by Benjamin Wilde in a design studio course taught by Prof. Rodney Hill at TAMU, Spring 2010. (Source: http://benwilde.blogspot.com/, Benjamin Wilde)
Figure 2
Parametric models created in Revit Architecture 2010 by Wei Yan for the lighting appliance. Top: two models of the same geometry but different materials. Note that the upper-right one’s material is of a curtain system. Lower-left: top view of the model. Lower-right: different parameters result in a different design.
The parametric design diagram and formulas (serving a part of a parametric modeling proposal) can be seen from Figure 3.

![Diagram with parameters and equations](image)

**Figure 3**
Parametric modeling proposal with a diagram, parameters, equations, and solutions (formulas), drafted by Wei Yan.
Method 1 – UI with Database

The design can be implemented in Revit Family Editor and MS Access with VBA Macro (Figure 4). Note: Revit doesn’t provide a way to store the family data like the accumulated sum of angles of all current and previous units. That is why the Access is used for doing this calculation and updating the model.

![Parametric modeling implementation in Revit Family Editor, MS Access with VBA macro, implemented by Wei Yan.](image)
Method 2 – API Programming

Alternatively, the entire creation of the geometry from scratch can be done in Revit API. The result and source code are given below. Note the same calculation is used in the source code. The program uses a recursive function to create the multiple levels of the square. It also assigns a material (glass) to the geometry, to demonstrate one more BIM component of the process (Figure 5).

![Parametric modeling implementation in Revit API (Wei Yan).](image-url)


Revit API source code

```csharp
/// <summary>
/// A class inherits IExternalCommand interface.
/// Recursive Squares, created by Wei Yan, Ph.D., Texas A&M University, Spring 2010
/// </summary>
public class RecursiveSquare : IExternalCommand
{
    /// <summary>
    /// Implement this method as an external command for Revit.
    /// </summary>
    /// <param name="commandData">An object that is passed to the external application which contains data related to the command, such as the application object and active view.</param>
    /// <param name="message">A message that can be set by the external application</param>
```
which will be displayed if a failure or cancellation is returned by
the external command.</param>

A set of elements to which the external application can add elements that are to be highlighted in case of failure or
cancellation.</param>

/// <returns>Return the status of the external command.
/// A result of Succeeded means that the API external method functioned as expected.
/// Cancelled can be used to signify that the user cancelled the external operation at some point. Failure should be returned if the application is unable to proceed
with the operation.</returns>

Document doc;

public Result Execute(ExternalCommandData commandData, ref string message, ElementSet elements)
{
  app = commandData.Application.Application;

  Transaction transaction = new Transaction(doc, "Position");
  transaction.Start(); // Start a transaction to allow changes to the model

  FamilyManager fm = doc.FamilyManager;
  FamilyParameter paramDepth;
  int depth = 10; //default value;
  if (paramDepth != null)
  {
    FamilyType t = fm.CurrentType;
    depth = (int)t.AsInteger(paramDepth); //specific type is needed for getting parameter value,
  }

  Material pMat = findElement(BuiltInCategory.OST_Materials, "Glass") as Material;
  FamilyParameter famParamFinish;
  famParamFinish = fm.get_Parameter("SquareFinish");
  if (famParamFinish == null)
  {
    famParamFinish = fm.AddParameter("SquareFinish", ParameterType.Material, true);
    if (pMat != null)
    {
      ElementId idMat = pMat.Id;
      fm.Set(famParamFinish, idMat);
    }
  }

  // in Revit, manually delete the old instances before running this program to create new ones.
  CreateSquare(depth, 0, famParamFinish);
  transaction.Commit();
  return Result.Succeeded;
}

private void CreateSquare(int depth, double angle, FamilyParameter famParamFinish)
double hd = depth + 5; // half of d
double d = 2 * hd;
double e = 1;
double levelHeight = 2;
XYZ p1, p2, p3, p4;
double z = (25 - depth) * levelHeight;
p1 = new XYZ(0 + hd, 0 + hd, z);
p2 = new XYZ(0 - hd, 0 + hd, z);
p3 = new XYZ(0 - hd, 0 - hd, z);
p4 = new XYZ(0 + hd, 0 - hd, z);

// Create extrusion profile
ReferenceArray ref_ar = new ReferenceArray();

rparray.Append(doc.FamilyCreate.NewReferencePoint(p1));
CurveByPoints cbp1 = doc.FamilyCreate.NewCurveByPoints(rparray);

rparray.Clear();
rparray.Append(doc.FamilyCreate.NewReferencePoint(p2));
CurveByPoints cbp2 = doc.FamilyCreate.NewCurveByPoints(rparray);

rparray.Clear();
rparray.Append(doc.FamilyCreate.NewReferencePoint(p3));
CurveByPoints cbp3 = doc.FamilyCreate.NewCurveByPoints(rparray);

rparray.Clear();
rparray.Append(doc.FamilyCreate.NewReferencePoint(p4));
CurveByPoints cbp4 = doc.FamilyCreate.NewCurveByPoints(rparray);

//Rotate to the right angle
Autodesk.Revit.DB.Line axis = doc.Application.Create.NewLineBound(new XYZ(0, 0, 0),
new XYZ(0, 0, 1));
doc.Rotate(cbp1, axis, angle);
doc.Rotate(cbp2, axis, angle);
doc.Rotate(cbp3, axis, angle);
doc.Rotate(cbp4, axis, angle);

ref_ar.Append(cbp1.GeometryCurve.Reference);
ref_ar.Append(cbp2.GeometryCurve.Reference);
ref_ar.Append(cbp3.GeometryCurve.Reference);
ref_ar.Append(cbp4.GeometryCurve.Reference);

// Extrusion based on direction and profile
Autodesk.Revit.DB.XYZ direction = new Autodesk.Revit.DB.XYZ(0, 0, levelHeight);
Autodesk.Revit.DB.Form form = doc.FamilyCreate.NewExtrusionForm(true, ref_ar, direction);

//Material can be changed from UI, demonstrating the interaction between UI and API.
Autodesk.Revit.DB.Parameter paramMat = form.get_Parameter("Material");
doc_FAMILYMANAGER_ASSOCIATEELEMENTPARAMETERTOFAMILYPARAMETER(paramMat, famParamFinish);

if (depth > 2) // depth is long enough to have children …
{
    // See Ben's Project API ReadMe.doc for the math.
    angle = Math.Atan((d / 2 - Math.Sqrt(8 * e * e - 8 * d * e + d * d) / 2) / (d / 2 + Math.Sqrt(8 * e * e - 8 * d * e + d * d) / 2)) + angle;
    CreateSquare(depth - 1, angle, famParamFinish); //Recursively create squares.
}

//=================================================
//   Helper function inspired by Jeremy Tammik's rfa_labs_20091028
Element findElement(BuiltInCategory bic, string targetName)
{
    // get ready to filter across an entire document
    FilteredElementCollector coll = new FilteredElementCollector(doc);
    List<Element> elems = coll.ToList();
    foreach (Element elem in elems)
    {
        if (elem.Name.Equals(targetName)) // we found it. return this.
        {
            return elem;
        }
    }
    // cannot find it.
    return null;
}