

# REALTIME **SPLINE** **EXTRUSION** MODELING FOR **3D PRINTING**

DEVELOPING A USER FRIENDLY CUSTOMIZER

BRIAN KEHRER, CREATIVE TECHNOLOGIST, PSYOP  
FOUNDER, ICOSAHEDRA

# THE PROBLEM

THE AWESOME PART ABOUT 3D  
PRINTING IS **PER-USER**  
CUSTOMIZED OBJECTS



USERS ARE ***TERRIBLE*** AT  
DRAWING, *FORGET* **3D**  
**MODELING**

‘CUSTOMIZER’ SERVICES ARE TOO  
VARIED, AND RESULT IN **USER  
CONFUSION**

WE NEED TO **ENABLE USER  
AGENCY**, WHILE BOUNDING  
THE OUTCOMES



# **MY APPROACH**

MY APPROACH

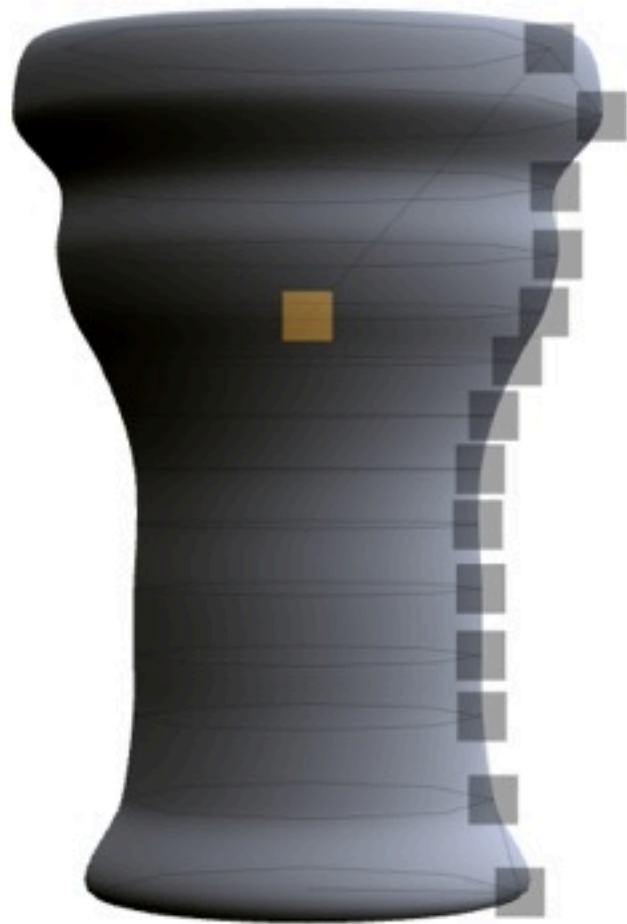
# **SPECIFIC** USECASES



# ORTHOGONAL TOOLS

# **FRIENDLY** INTERFACE

MY APPROACH



THINGIVERSE HACK  
~ 12 HOURS OF DEVELOPMENT



# ***THINGIVERSE HACK***

- CATMULL-ROM SPLINES
- FIXED TESSELATION
- EXPORT TO STL
- UPLOAD TO THINGIVERSE
- RUNS ON IOS & ANDROID

# **BÉZIER SPLINES**

# ***WHAT IS A SPLINE?***

A SPLINE IS A SERIES OF CURVES, WITH THE  
**END POINTS CONNECTED**, AND,  
OPTIONALLY, **CONTINUOUS  
TANGENTS**.



# LOTS OF SPLINES

ALL USING THE SAME MATH

- CATMULL-ROM - **USER DEFINED POINTS ON CURVE**
- QUADRATIC BÉZIER - **CLOSED FORM SOLUTIONS**
- CUBIC BÉZIER - **COMMON IN ART SOFTWARE**
- KOCHANER-BARTELS - **UNIQUE PARAMETERS**

```
public class CatmullRom2D : ISpline2D{  
  
    Vector2[] points;  
  
    public Vector2 EvaluatePosition(float t){  
        //the math is on wikipedia  
    }  
    public Vector2 EvaluateDerivative(float t){  
        //dy/dt the math above  
    }  
}
```

```
public static class SplineMath{  
    public static Vector2 EvaluatePosition(  
                                                Vector2[] points,  
                                                float t  
                                                ){  
        //the math is on wikipedia  
    }  
    public static Vector2 EvaluateDerivative(  
                                                Vector2[] points,  
                                                float t  
                                                ){  
        //dy/dt the math above  
    }  
}
```



```
public class CatmullRom2D : ISpline2D{

    private Vector2 EvaluatePosition(float t, int initialPoint){

        Vector2 previousPoint = (initialPoint == 0) ?
points[initialPoint] : points[initialPoint-1];

        Vector2 tangent1 = 0.5f*(points[initialPoint+1] -
previousPoint);

        Vector2 nextPoint = (initialPoint+2 < points.Length) ?
points[initialPoint+2] : points[initialPoint+1];

        Vector2 tangent2 = 0.5f*(nextPoint - points[initialPoint]);

        float tSquared = t*t;
        float tCubed = t*tSquared;

        return
            (2*tCubed - 3*tSquared + 1) * points[initialPoint] +
            (tCubed - 2*tSquared + t) * tangent1 +
            (-2*tCubed + 3*tSquared) * points[initialPoint+1] +
            (tCubed - tSquared) * tangent2;
    }

}
```

```
void PositionVertices(){  
    for(int i=0; i<rings; i++){  
        for(int j=0; j<slices; j++){  
            float t = Mathf.PI * 2f * ((float)j/(float)slices);  
            float percent = (float)i / ((float)rings-1);  
            Vector2 pos = spline.EvaluateSplinePosition(percent);  
            float xPosition = pos.x*Mathf.Sin(t);  
            float zPosition = pos.x*Mathf.Cos(t);  
            int idx = j + i*slices;  
            vertices[idx] = new Vector3(xPosition,pos.y,zPosition);  
        }  
    }  
}
```



# ***PRETTY BASIC...***

YES, BY DESIGN. BECAUSE IT'S **SO SIMPLE**,  
WE SHOULD HAVE NO TROUBLE  
GENERATING **EQUAL AREA UV**  
COORDINATES, WHICH ALLOWS FOR  
ADDITIONAL FEATURES, LIKE **LUMINANCE**  
**BASED DISPLACEMENT.**



# CONSTRAINTS

USING THE **SAME**  
**UNDERLYING CODE**, WE CAN  
DESIGN DIFFERENT CUSTOMIZERS  
FOR DIFFERENT PRODUCTS BY  
**IMPOSING CONSTRAINTS.**

# ***SIMPLICITY***

## **FROM CONSTRAINTS**

- MATERIAL PROPERTIES
- ARCHETYPAL FORMS
- HUMAN LIMITS & SCALE
- COMMON UNITS OF MEASURE
- USER-FLOW



# CERAMICS APP

DESIGN COMMON **HOUSEHOLD CERAMICS** ON A MOBILE DEVICE, CUSTOMIZED TO YOUR OWN LIKING, AND HAVE THEM **3D PRINTED**, AND **DELIVERED**.

# ***CERAMICS***

## **SPECIFIC CONSTRAINTS**

- CLAY WALL SIZE
- COMMON MUG SIZES (2<sub>oz</sub>, 8<sub>oz</sub>, 12<sub>oz</sub>, ETC)
- KILN LIMITATIONS (NO AIR POCKETS)
- GLAZE REQUIREMENTS
- USER-FLOW & PRODUCT ARCHETYPES



# ***NEXT STEPS***

LEVERAGE **EXISTING PRODUCT TAXONOMIES**, AND BUILD SPECIFIC CUSTOMIZATION ENGINES AROUND THOSE ARCHETYPAL PRODUCTS



THANKS  
**@BIRDIMUS**