Fractal Curves for Tool Path Planning

Darshan Makwana
Mechanical Engineering
Indian Institute of Technology, Bombay
Mumbai, Maharashtra
21d100003@iitb.ac.in

***Abstract*—Space-filling curves, like the Hilbert or Peano curves, are continuous fractal curves that can fill an entire space or plane without any gaps or overlaps. These curves are particularly interesting for tool path generation in additive manufacturing because they can help in creating uniform paths that can cover an entire print layer without lifting the tool, thus optimizing the manufacturing process. In this paper we formulate an algorithm that can fill an entire cross section using the hilbert curve. We then discuss about it's applications and other future work**

***Index Terms*—Hilbert Curve, Layered Manufacturing**

Link to the Github repo containing all the code developed for this term paper: https://github.com/darshanmakwana412/fractal

I. Introduction

In recent years layered fabrication is extensively used for various manufacturing purposes because it allows both for the creation of complex and intricate structures that would be difficult or impossible to achieve using traditional manufacturing methods. However Traditional methods for layered fabrication consists of designing a tool path that is niether continuous nor differentiable. This results in switching on and off the extruder a number of times resulting in uneven filament extrusion and overall affects the quality of the layer. In this paper we discuss an approach that leverages hilbert curves to fill an entire cross section

II. Related Works

The domain of tool path planning algorithms is very profound and there has been significant progress in the field in the recent years. I will share some of thoughts and basic underlying from some of the papers that I came across during my term paper Haisen Zhao and Fanglin Gu discusses the idea of connected fermat spirals for layered fabrication. They use spiral as their main choise of fractals. One interesting property about spirals is that they can fill a region by predefing the start and end points in the domain. The allows the path to very compact as well as smooth resulting in much greater travel speed of the tool while printing. Their method also recursely builds a tree of sprials and connecting the paths of spiral to each other is the tricky part. However their method fails to generalize when we bring in fractal curves.

Jordan Cox Initially explored the idea of using fractal curves for tool path planning but did not used the semantic understanding in the image maps. An instead attempted to perform a transformtion on the hilbert curve to fit inside the image. Obviously this method works only for a few select regions and even then does work as expected.

III. Methodology

In what follows next, we discuss our approach for travesing a hilbert curve inside a region $\mathcal{P} \subset R^2$ but in general any fractal curve can be used with minor modification to the decomposition algorithm which is described below. We first discuss how we can decompose the image into rectangular regions of differennt sizes while maintaining the semantic knowlege inside the image. We then use the rectangular deompositions to iteratively construct the hilbert curve in a continuous fashion

A. Rectangular Decomposition

Let us define Q_{11}(P) to be the top left quadrant of $\mathcal{P}$, similarly let Q_{12}(P), Q_{21}(P) and Q_{22}(P) be the top right, bottom left and bottom right quadrants of $\mathcal{P}$. Let also define $M(P)$ to be the mean value of all the pixel intensity values inside $\mathcal{P}$. If $M(P)$ is less than some threshold value P_{th} then we decompose $\mathcal{P}$ into $Q_{11}(P)$, $Q_{12}(P)$, $Q_{21}(P)$ and $Q_{22}(P)$, for each of them again we check if $Q_{ij}(P) < P_{th}$. Suppose $Q_{i^*j^*}(P) < P_{th}$ then we again decompose it into $Q_{11}(Q_{i^*j^*}(P))$, $Q_{12}(Q_{i^*j^*}(P))$, $Q_{21}(Q_{i^*j^*}(P))$ and $Q_{22}(Q_{i^*j^*}(P))$. We perform this procedure until either a fixed number of iterations denoted by N in the program below has been passed or the image can no longer be decomposed.

We thus keep on iteratively decomposing the image into smaller rectangular subregions based on the local image pixel intensity value inside the quadrants Below are the rectangular decompositions of an image for 6, 7, 8 and 9 iterations respectively

The above algorithm can be used for other curves such dragon cruve by usign triangles or the gosper curve by using hexagons

IV. Continuous Hilbert Fill

Having decomposed the region $\mathcal{P}$ into rectangular subregions, we now describe the procedure to aggregate them into continuous hilbert fills. We can do this by alternatingly joining the adjacent sides of the rectangles. To achieve this we rotate the curve where we met rectangle adjacent borders and invert the path if necessary. When we are inverting the order in which we visit Q_{11} and Q_{22} is flipped and both the regions are also rotated, and the direction of rotation is based on wether we

Algorithm 1 Rectangular Decomposition

```
function DECOMPOSE(i, x, y, quadrant)
    if quadrant == 0 then
        x ← 2 * x
        y ← 2 * y + 1
    else if quadrant == 1 then
        x ← 2 * x
        y ← 2 * y
    else if quadrant == 2 then
        x ← 2 * x + 1
        y ← 2 * y
    else if quadrant == 3 then
        x ← 2 * x + 1
        y ← 2 * y + 1
    end if
    if i < N-1 and i ≥ 0 then
        gray ← mean(P[N − i − i][x, y])
    else
        gray ← −1
    end if
    if gray ≥ 0 and gray < 256 - P_th then
        for q ∈ 0, 1, 2, 3 do hilbert(i+1, x, y, q)
        end for
    else
        print the rectangle with center (x, y)
    end if
end function
```

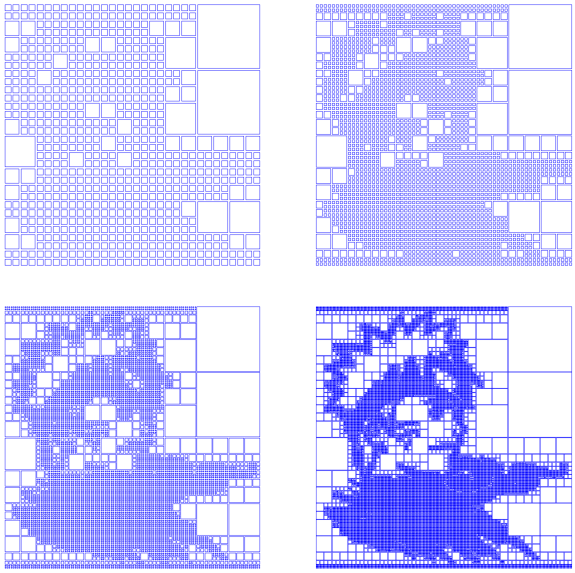

Fig. 1. Rectangular Decomposition for 6, 7, 8 and 9 iterations

have achieved an inversion in the previous quadrant or not. Now the rotation and inversion for each $Q_{i^*j^*}$ is computed as follows

Algorithm 2 Rotation and Inversion

```
if quadrant == 0 then
    if Inversion then
        rotation += -1
        Inversion = False
    else
        rotation += 1
        Inversion = True
    end if
else if quadrant == 1 then
    if Inversion then
        rotation += 1
        Inversion = False
    else
        rotation += -1
        Inversion = True
    end if
else if quadrant == 2 or quadrant == 3 then
    Do not change rotation and inversion
end if
```

We then visit a particular quadrant $Q_{ij}(Q_{i^*j^*})(P)$ with the rotation and inversion being defined above, and then traversing each quadrant specifically. It can be theoretically proved that this algorithm can fill the entire space

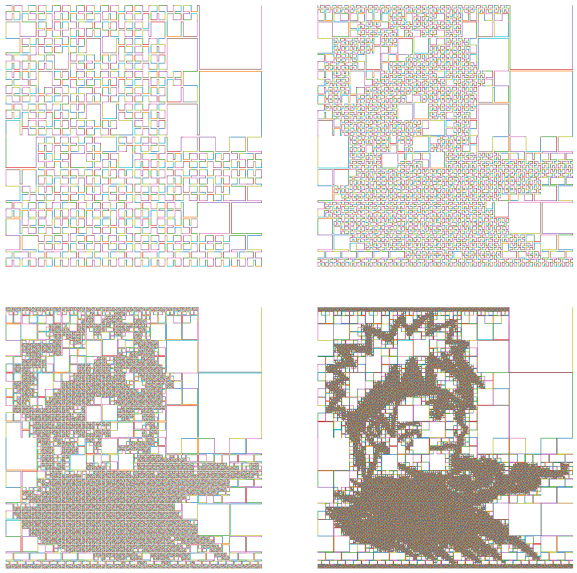

Fig. 2. Hilbert Fill for 6, 7, 8 and 9 iterations

V. Results and Optimizations

As this is a recursive implementation, since we are calling the function approx 4 million times for just 8 iterations a lot

of overhead gets called for calling the function. To reduce this I implemented a paralled implementation of the same function using numba which is a C++ accelerated implementation of numpy. This resulted in significant boost to the overall computation time and a lot of overhead was reduced Below are some of tool path planned using the above described algorithm

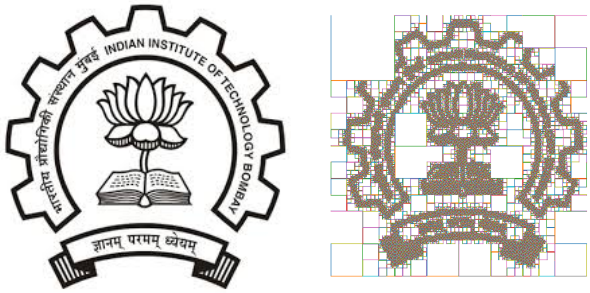

Fig. 3. Logo of IITB

VI. Future Work

Though we can add these paths in a layer wise fashion, We can also vary the thresholding of an img as we iteratively build these layers. This will result in beautiful patterns where some hilberts are smaller and hide beneath larger hilbert curves in a layer. An example of which is shown here

Fig. 4. Layer wise thresholding

Acknowledgment

To all the TA's for helping out setting up the 3d printer and allowing me to use it whereever necessary and to Prof. Gurminder Singh for allowing me to take up this exceptionally incredible project and also guiding me on how to start working on it

References

[1] Haisen Zhao and Fanglin Gu and Qi-Xing Huang and Jorge Garcia and Yong Chen and Changhe Tu and Bedrich Benes and Hao Zhang and Daniel Cohen-Or and Baoquan Chen, "Connected Fermat Spirals for Layered Fabrication", ACM Transactions on Graphics (Special Issue of SIGGRAPH), Article No. 100, 2016

[2] Adrien Bedel, Yoann Coudert-Osmont, Jonàs Martínez, Rahnuma Islam Nishat, Sue Whitesides, et al.. Closed space-filling curves with controlled orientation for 3D printing. Computer Graphics Forum, In press, ff10.1111/cgf.14488ff. ffhal-03185200v2f

[3] A. Bedel, Y. Coudert-Osmont, J. Martínez, R. I. Nishat, S. Whitesides and S. Lefebvre, Closed space-filling curves with controlled orientation for 3D printing, working paper or preprint (2021), https://hal.inria.fr/hal-03185200. Google Scholar

[4] M. Bertoldi, M. Yardimci, C. M. Pistor and S. I. Guceri , Domain decomposition and space filling curves in toolpath planning and generation, in Proc. Ann. Int. Solid Freeform Fabrication Symposium (SFF'98) (1998), pp. 267–276. Google Scholar

[5] B. Chazelle and D. Dobkin , Decomposing a polygon into its convex parts, in Proc. Eleventh Ann. ACM Symp. Theory of Computing, STOC '79 (1979), pp. 38–48, New York, USA, Association for Computing Machinery. Crossref, Google Scholar

[6] K. G. Jaya Christiyan, U. Chandrasekhar and K. Venkateswarlu , A study on the influence of process parameters on the mechanical properties of 3d printed ABS composite, IOP Conference Series: Materials Science and Engineering 114 (2016) 012109. Crossref, Google Scholar

[7] D. Ding, Z. Stephen Pan, D. Cuiuri and H. Li , A tool-path generation strategy for wire and arc additive manufacturing, The Int. J. Adv. Manuf. Technol. 73(1–4) (2014) 173–183. Crossref, Google Scholar