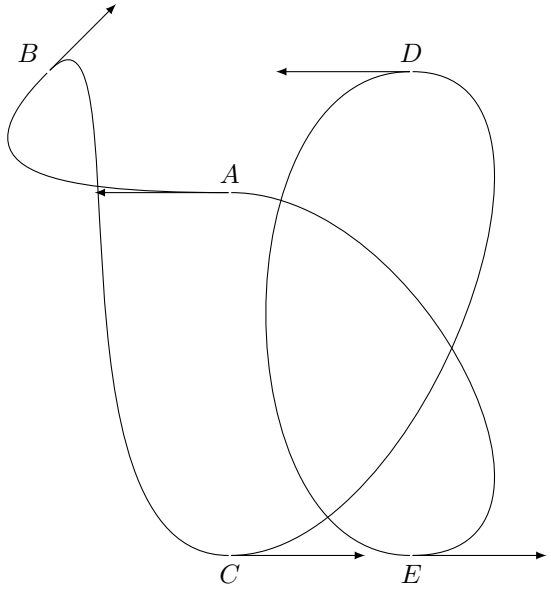
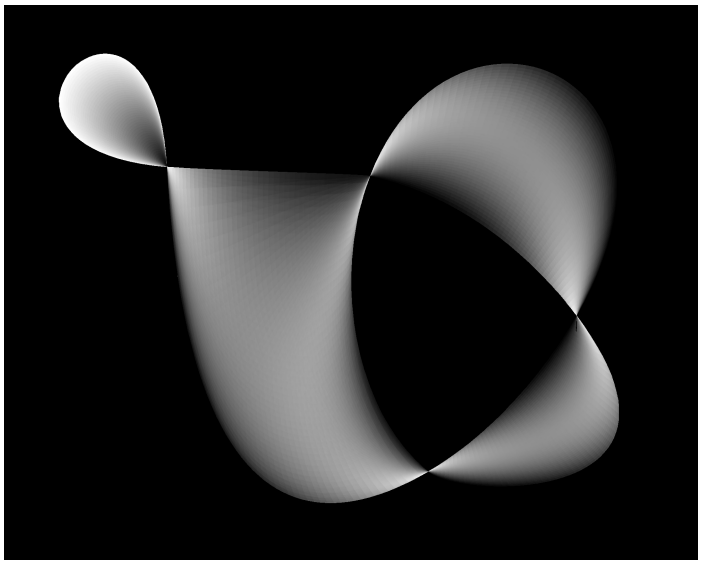
Theory on Vi Hart's Flippy Chips

Anton Obersteiner

November 19, 2020





1 Overview

Goal The Goal is to write a Program that can generate Images with a Coloring that has a nice 3D-flipping Look from a closed Line as drawn by YouTuber and Mathematician Vi Hart. I'll call the result a 'Chip'.

Challenges To achieve that, a few mathematical and computational Challenges need to be overcome, those are mainly:

- Finding the Intersections of the Line
- Generating the Graph of the Chip
- Defining the value of the Chip in the Faces

Behind the #Includes A lot has to be done behind the Curtains of a few `#include` statements. In the Area of simple and composed Splines:

- Defining and Implementing Splines
- Finding Intersections of a Spline with another and with itself
- Finding Intersections of a Spline with a straight Line
- Getting a part of a Spline as a new Spline
- Implementing almost all of these for Spline Constructs (several Splines) too
- Approximating Spline Constructs with one single Spline

And for actually Creating, Composing and Manipulating the resulting Images in a memory-efficient way:

- Defining Canvas Classes
- α-Channels and Layered Canvasses
- Drawing Basic Shapes (when Lines are the hardest thing)
- Filtering (smooooth Gauss!)

Outlook The Project has resulted in a Prototype, not a full and final Program, many things are glued together behind the scenes and would fall apart on other Examples of a Chip.

2 Splines

The Line that is the basis for a Chip is made from a Series of Splines. Splines are functions that take a number from 0 to 1 and return a smoothly changing 2D-Value from a Startpoint P to an Endpoint Q. They start at P going out with a speed and direction defined by p and enter Q following q. I'll either write them as $\mathrm{L}_{PQpq}(t)$ or simply as $\widetilde{PQ}(t)$ if p and q are inferable from context.

$$\begin{aligned}\mathrm{L}_{PQpq}(t) =& P + pt + (3Q - 3P - 2p - q)t^2 + (2P - 2Q + p + q)t^3\\ &\Rightarrow \mathrm{dL}(0) = p, \qquad \mathrm{dL}(1) = q\end{aligned}$$

The following abbreviation will be useful for readability purposes:

$$\begin{aligned}T &:= 3Q - 3P - 2p - q\\ U &:= 2P - 2Q + p + q\\ \Rightarrow \mathrm{L}_{PQpq}(t) &= P + pt + Tt^2 + U^3\end{aligned}$$

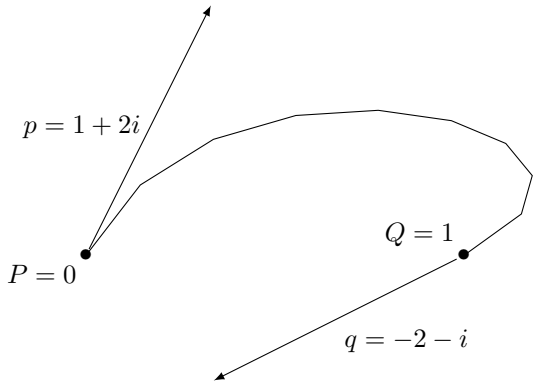


Figure 1: Spline $\mathrm{L}_{0,1,1+2i,-2-i}$

diff:

$$\mathrm{dL}_{PQpq}(t) = p + 2Tt + 3Ut^2$$

subspline: The Spline following L from t_1 to t_2 can be defined as follows:

$$\begin{aligned} P &= \mathrm{L}(t_1) & Q &= \mathrm{L}(t_2) \\ p &= (t_2 - t_1)\mathrm{dL}(t_1) & q &= (t_2 - t_1)\mathrm{dL}(t_2 - t_1)(t_2)) \end{aligned}$$

2.1 Intersection of Splines

Finding out, that a Solution for the Intersection $\mathrm{L}_1(t) = \mathrm{L}_2(u)$ is increadibly difficult and generally impossible to find analyically, took a long time and a lot of lines of LaTeX... When I realized that I could not do it, I went with the unclean, imprecise and much less satisfying approach of putting hairbands on the Splines everywhere and then letting them slowly wiggle together. What I'm talking about is Gradient Descent, with a number of Parameters t along Spline A and the same number of Parameters u along B. Every possible Pair of Parameters (t, u) is slowly changed to reduce the Distance of the Points $A(t), B(u)$ according to a very simple Approximation of the Gradient:

$$\begin{aligned} \mathrm{d}t =& \frac{\alpha}{2\varepsilon} \left(|A(t - \varepsilon) - B(u)| - |A(t + \varepsilon) - B(u)| \right) \\ \mathrm{d}u =& \frac{\alpha}{2\varepsilon} \left(|A(t) - B(u - \varepsilon)| - |A(t) - B(u + \varepsilon)| \right) \end{aligned}$$

Here, α is the speed of the Approximation, and there must be a hundred ways to make the Convergence more efficient by changin it over time or a lot of other things. Additional measures that are used in the Algorithm:

- Accepting Pairs, where the Distance of the Points is less than `done_crit` (Default: 0.005), as Solutions and removing them from the active pool
- Removing Pairs, where t or u are outside the Spline's Definition $[0, 1]$
- Removing Pairs that are almost the same as an accepted Solution (t_2, u_2): $|A(t) - A(t_2)| <$ `same_crit` and $|B(u) - B(u_2)| <$ `same_crit` (Default: `same_crit` $= 0.02$)
- Removing Pairs that are not changing anymore ($|\mathrm{d}t|, |\mathrm{d}u| <$ `fixed_crit`, Default: `fixed_crit` $= 0.0001$)
- In Self-Intersection: $A = B$, so $t = u$ is useless and gets removed ($|t - u| <$ `identical_crit`, Default: 0.05)

Another measure is rather practical: Two Splines that are subsequent – or: Neighbors – in the Line, will always have the Intersection where they join. That Solution is of no value for the actual pupose and is therefore removed, if the corresponding Parameter `neighbors = true` is passed. Self-Intersection calls also have to be marked in a similar way, with `identical = true`.

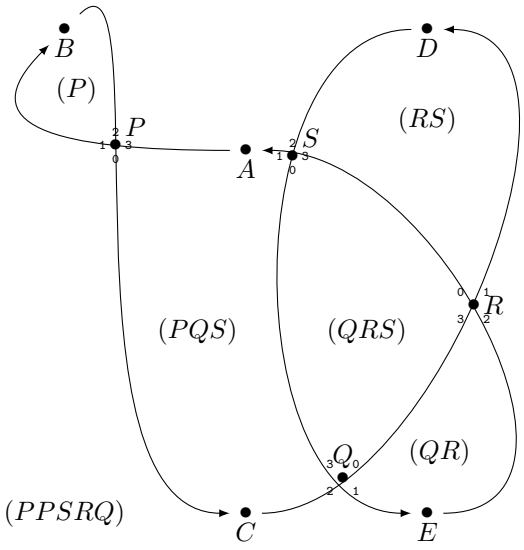


Figure 2: All Points $A, B, ...$ and Intersections/Nodes P, Q, R, S with Faces

3 Graph

Edges When all the Intersections $P, Q, R, ...$ in the Chip are found, that gives away neither the Edges nor the Faces immediately. For every Intersection, the Parameters t and u are given and also, more importantly, the Points A and C, they come from. with that information, an algorithm can follow the Line from Spline $\widetilde{AB}(t)$ on, until it finds another Intersection/Node (A_2, C_2, t_2, u_2). In the Example, P is roughly at $\widetilde{AB}(0.26)$. Then $\widetilde{AB}(.26)$ is followed to B and then from $\widetilde{BC}(0)$ to $\widetilde{BC}(.39)$, which is also P.

Faces Numbering the outgoing Edges of a Node P as `P0, P1, P2, P3` clockwise gives us then the ability to follow along the inside of a Face. `P0` leads to `Q2`, then we follow `Q3 -- S0`, then `S1 -- P3` and we would next follow `P0` so we have found the Face (PQS). In the Example, this method yields Faces $(PQS), (QRS), (QR), (RS), (P)$ and the Outside Face $(PPSRQ)$.

Inside − Outside Which Face actually is the Outside Face, is rather important because only the Inside Faces will be colored. The Algorithm detects the Outside Face by calculating the first Intersections of any Edge with the Straight Line from $(0, 0)$ to $(1, 1)$. This isn't perfect but a more general definition isn't needed currently. Once the Outside is known, all adjacent Faces are inside. Vice versa and so on until all Faces are either Outside or Inside. This makes (QRS) Outside for example.

4 Net

To finally color a Face, I define a Net of Splines going either out from every Corner or in waves around every Corner. In the Example below, the Corner is P with Q on one and R on the other side. So, one line is going out from P to Q along the Spline that connects them, and one from P to R:

$$\begin{aligned} \mathrm{F}(t, 0) &= \mathrm{L}_{P,Q,p_1,q_2}(t) \\ \mathrm{F}(t, 1) &= \mathrm{L}_{P,R,-p_2,-r_1}(t) \end{aligned}$$

Then, the rest are Interpolated between those two: The Begin stays at P, the First Control (p) changes from p_1 (leads towards Q) to $-p_2$ (to R) with a Parameter v: I call that direction $c(v)$. The End Point moves from Q to R

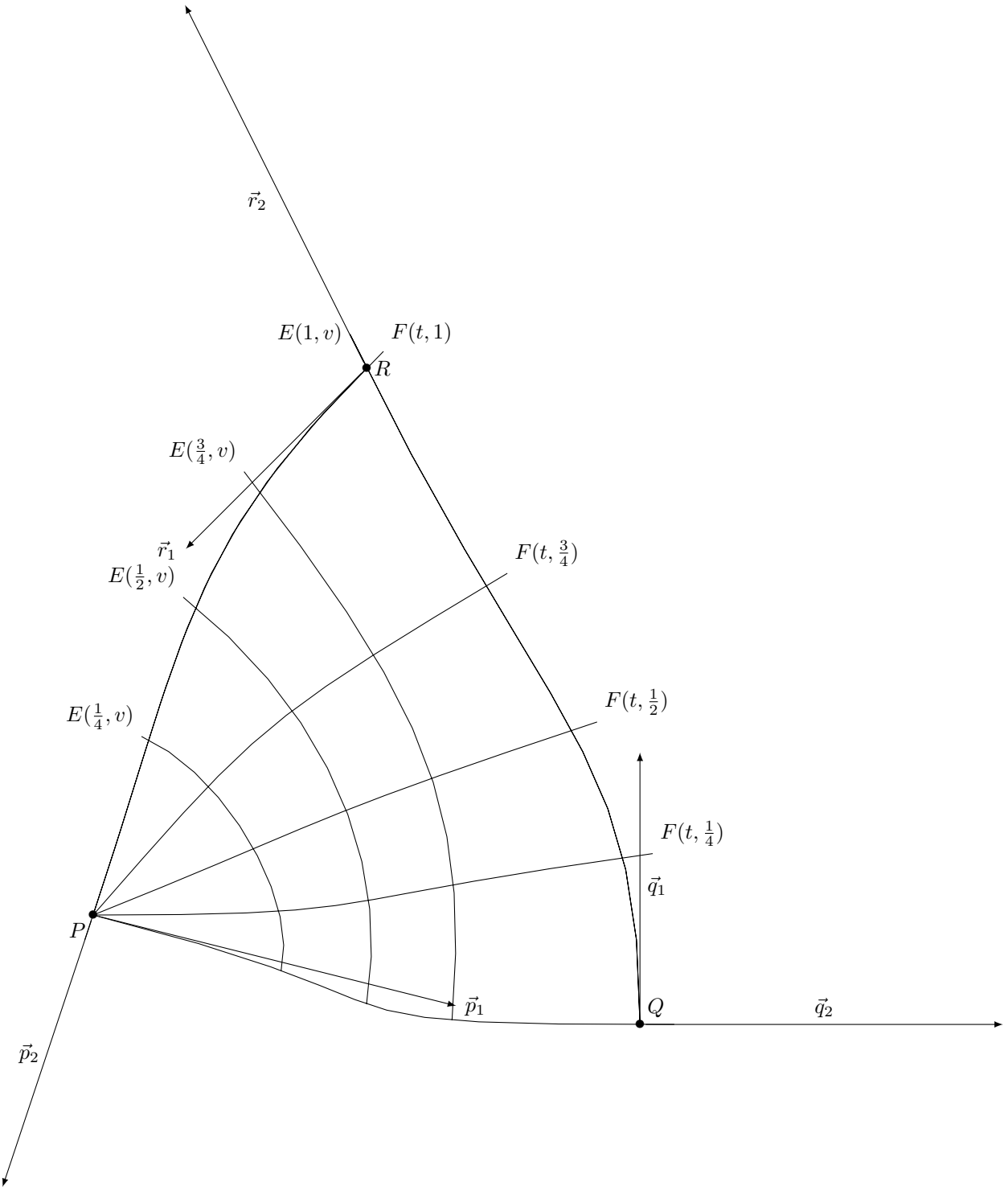


Figure 3: Net of $\mathrm{E}_{PQRp_1p_2q_1q_2r_1r_2}(t,v)$ and $\mathrm{F}_{PQRp_1p_2q_1q_2r_1r_2}(t,v)$

along the Spline defined by the other directions of Q and R, those that don't lead towards P: q_1, r_2: That's $C(v)$. And the direction of the Spline at the End Point is Interpolated between the direction of $\widetilde{PQ}$ at the End and $\widetilde{PR}$ at the End: $d(v)$. It is important to use Interpolated directions and not just Right Angles here, because near the actual Corners, they need to match the Edges' Angles, which don't got out neccessarily perpendicularily.

$$\begin{aligned} C(v) &:= \mathrm{L}_{QRq_1r_2}(v) \\ c(v) &:= -p_2 v + p_1(1-v) \\ d(v) &:= -r_1 v + q_2(1-v) \\ \mathrm{F}(t,v) &= \mathrm{L}_{P,C(v),c(v),d(v)}(t) \end{aligned}$$

$\mathrm{E}(t,v)$ is defined similarly, but the Splines go from $\widetilde{PQ}$ to $\widetilde{PR}$ and not along these, they move out in waves and not in beams. It's Implementation is currently faulty and I have not used it because F was sufficient for me, but it could be worth looking into.

$$\begin{aligned} A(t) &:= \mathrm{L}_{PQp_1q_2}(t) \\ B(t) &:= \mathrm{L}_{PR-p_2-r_1}(t) \\ a(t) &:= q_1 t - p_2(1-t) \\ b(t) &:= r_2 t - p_1(1-t) \\ \mathrm{E}(t,v) &= \mathrm{L}_{A(t),B(t),a(t),b(t)}(v) \end{aligned}$$

When there is a structure to every Corner of a Face, the only thing missing is the correct Color assignment for the little Quadritalerals...

5 Value

The Value $\eta(t,v)$ that is to be assigned depending on t and v in a Corner of a Face must start with white on one side out from P and black on the other. (Which makes it undefined at P itself, but that is a problem for Theoreticians to deal with...) It must also end in white everwhere else and slowly change from one Edge $\widetilde{PQ}$ to the other $\widetilde{PR}$ between the Values found there. And it should get more and Transparent as it gets further away from the Corner Node itself (P), because it is only "competent" near P and can't tell us anything about the environment near Q or R. The opposite of Transparency is either corruption or Opacity and for Opacity, the symbol α is common. I have chosen the following Formulae for my Program:

$$\begin{aligned} \eta(t,v) &:= (t^3(1-v)+v) \\ \alpha(t,v) &:= (1 - 0.99(t^3(1-v)+v)) \end{aligned}$$

The 0.99 just ensures, that the Opacity never is 0, so there is "always some color there", in case it is needed for lack of another Corner to "overcolor" it.

6 Outlook

Generalization There are a lot of details still not fully generalized:

- The Outer Face is detected with a diagonal, but that could also miss the Chip completely.
- The Coloring only uses regular Splines, but usually the Edges are composed of at least two Splines, so there will be only an Approximation and for many cases of Lines, that would look terrible. To see this happen, you can look at the little ear in the top left corner of my First Page.
- With more than three Edges to a Face, it could easily happen that the innermost Area is not covered by any Net and therefore stays opaque, though it could just be white. That **would** be awkward.

Optimization The next Challenge will be to make the algorithm more effient, especially when redrawing a Chip with a slightly changed Point. The Canvas Classes could probably also take some help from CUDA to become more time efficient – though I already made them as memory efficient as neccessary for large images on my machine.

Interactivity Also, although there are some pieces of old code in `test.cpp` that use the mouse, these are made for the annoyingly buggy `graphics.h` which always took my `X` with it when it crashed – sometimes also, when the program just terminated normally – so there is no Interactivity at all currently, even not using the commandline, the Points for the Line are just coded in. And I couldn't learn OpenGL in the last 5 weeks...