Week 10 Report (8/11/16)

Changes Added

Rotation has been fixed and implemented as intended. A rotation box has been added to represent orientation of the element. The top of the box is colored to differentiate what orientation the element is in. If any of the vertices and dragged on, an angle is calculated and set onto that element. Rotation can be seen with saddle element as it maintains its cross orientation. Attachment and Separated elements look more visually correct due to changes in weight values.

Element Interaction And Weights

Interaction between singular elements have changed after incorporating rotation and scaling matrices. I believe this might have been a problem with normalization and I have fix it. I will to continue to check if there are other things I have missed.

There are many different types of weights in this vector field that determines the interaction between the elements. The different types of weights can affect the influence or size of the element and the interaction and blending between each of the elements. Ideally, the vectors should blend with each other and now show a sharp vector change. Playing around with the numbers should get the visuals fine-tuned. There is still more fine turning needed to match weight interaction between different type of elements to blend better. However, many other factors can change for the vector field and it will make it hard to recalibrate the weight values. So storing user input weight values will allow easier comparisons. The saved weight values will be done through reading and writing save files.

B(x) for Attachment/Divergent Elements –

weight = (C\***Math**.*exp*(-k\*distance\*distance));

C determines the element’s span onto the vector field

K determines the “sharpness” of the element and how sharp the vectors converge or diverge

Line Integral Convolution

|  |
| --- |
|  |

From what I have read, we can start with arrow plots that then help determine flow direction but not the velocity because the field is normalized. A vector field streamline will then be generated from each point of the flow arrows. This process is called integration and it creates the represented streamline. By stepping forward and backwards from the differential, we can draw the line. From there, we will add a white noise texture onto the flow field to generate the pattern that represents a LIC image by using an image kernel technique. The location of the streamline is mapped from a mesh where the texture coordinates can also be plotted. Below is pseudocode for the streamline and integration.

interface Differentiable {

vec\_at(x: number, y: number, v?: Vec2): Vec2;

}

interface Integrator {

// Step size parameter

public stepSize: number;

// Differential calculator function

public diff: Differentiable;

step(x: number, y: number): Point;

stepReverse(x: number, y: number): Point;

}

class Euler implements Integrator {

public v: Vec2;

constructor(public stepSize: number, public diff: Differentiable) {

this.v = new Vec2(0, 0);

}

step(x: number, y: number): Point {

this.v = this.diff.vec\_at(x, y, this.v);

return new Point(x + this.v.x, y + this.v.y);

}

stepReverse(x: number, y: number): Point {

this.v = this.diff.vec\_at(x, y, this.v);

return new Point(x - this.v.x, y - this.v.y);

}

}