Algorithms and Process Report

For my Programming assignment I choose to write a math driven Tornado Creator.

I started by writing my Tornado curve function as this was what calculates the main Tornado movement.

I visualized the outcome with MatLab.

The calculation of the tornado curve consist of 3 different steps:

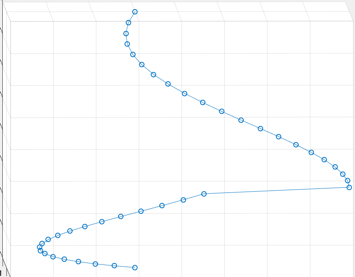
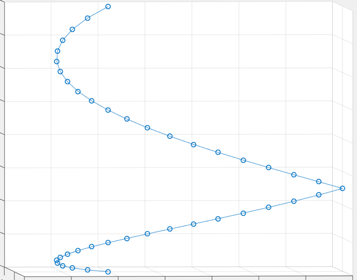
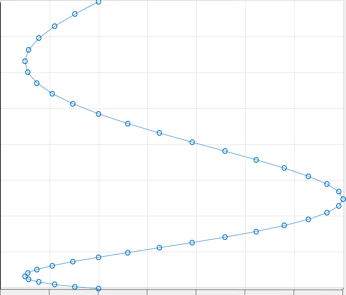
Calculate guide curves by interpolating between a start value an end value and a control point.

Calculate a middle Curve by interpolating by the previously calculated guide curves. This is going to create a movement of the tornado.

I looked at different interpolation methods

The best one turned out to be cubic interpolation

Linear interpolation Quadratic Interpolation Cubic interpolation



Taken a point from the middle curve create a circle around it. By moving up the middle curve over time this creates a spiral. This spiral is also manipulated by some more attributes to create a different more natural tornado movement movement.

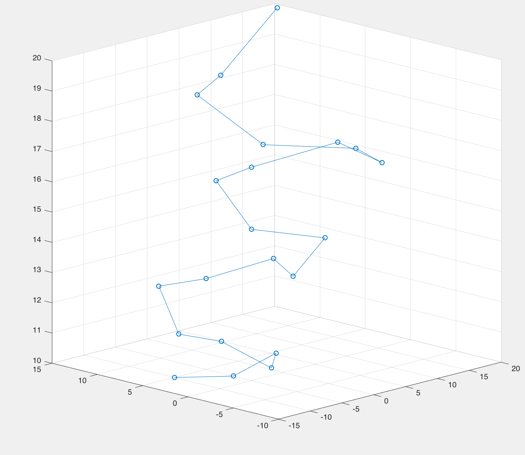
X= o(tx) + r\*(time+startValue) \* sin(speed\*time+ offset value)

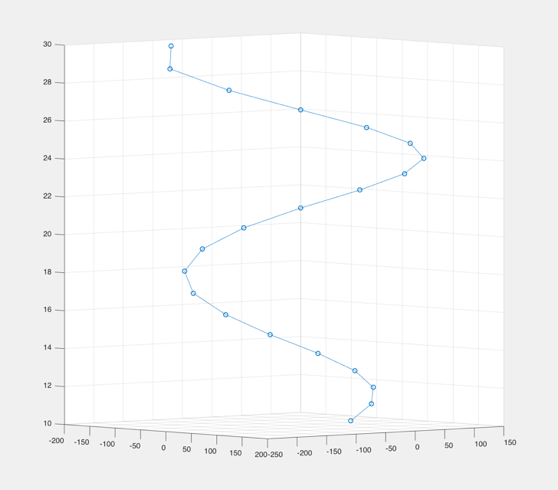
Y= o(ty) + r\*(time+startValue) \* cos(speed\*time+ offset value)

Z= o(tz) + speed\*time

Z is representing the up axis.

The o value is calculated by the middle curve and than an adjusted sin or cosine value is added. Multiplying the radius by a time value will increase the radius as the tornado goes upwards. The sine and cosine functions are used with a product of speed and time which lets us have control about how fast the tornado rotates.

High speed Less speed 

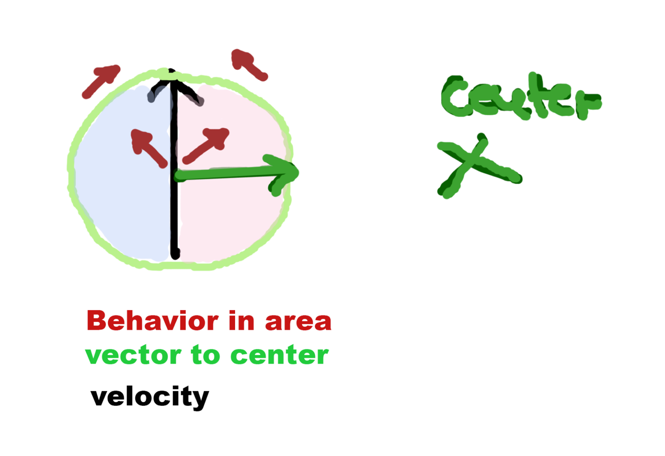
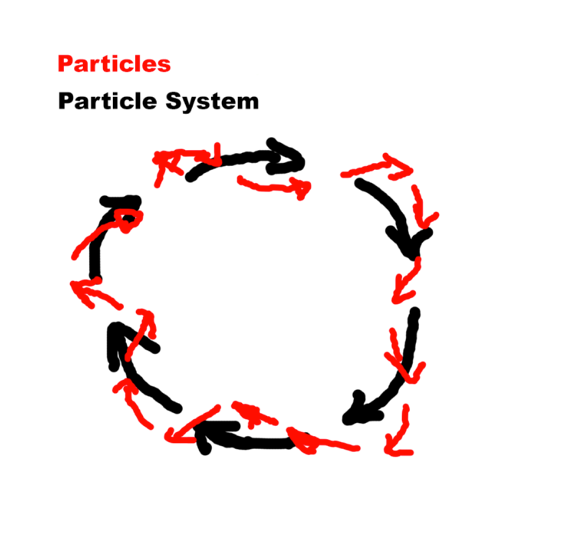


As the tornado Curve Class does most calculations the tornado class and particle system class are mainly organizing the tornado rather than actually calculating anything.

I decided to use std::vector for storing the particle systems and particles as it was the most efficient and flexible option.

For the Particle movement I allowed the user to choose a type of motion.

One option is an algorithm that takes into account the tornado center and create an inwards outwards flow of the particles simulating the actual inwards outwards drift in a tornado



To achieve this movement by this algorithm

if the particle is outside of the particle system bounding box:

particle velocity = (0.3\*velocity)+(0.5\*velocity of the particle system )+(0.2\*vector to the particle center)

else if the particles are in a buffering area:

(this is their so the transition between the to areas is smoother)

velocity will stay the same

else if it on the inside of the tornado:

(velocity goes outwards)

velocity=(0.5\*velocity)+(0.3\* velocity of the particle system) + (0.2\*vector to the tornado center)

else if its on the outside:

(velocity goes inwards)

velocity= (0.5\*velocity)+(0.3\* velocity of the particle system) - (0.2\*vector to the tornado center)

Otherwise, if the user selects “Random Points”, the algorithm used for the particle movement is the random points algorithm where the particle is placed in a random position inside the bounding box every frame.

The original idea was to make it move from one point to the next but changing the point every frame produced a better movement. This is imitating the idea of fuzzy objects mentioned in my initial program design.

Talk about

Lifetimes

Reaching seeling

Pick up

Shaders: alpha / texture

Update

UML diagrams