David Grzesik

VSFX – 350 Procedural Modeling And Animation

Deborah Fowler

Project 1 – Complex Scene

Reaction Diffusion

Render

1.75 Million Simulated Points - 5000 Frames of Simulation - 233 GB Cache - 10 Minute Image Render

This scene displays a later frame of a reaction diffusion simulation on a point cloud scattered in a 3D scan of a sculpture. The final render didn't come out as aesthetically pleasing as I wished it to be, mainly because too much time was spent on developing a system that properly replicated reaction diffusion in three dimensions than on lighting, materials, and rendering. I am, however, very satisfied with the success of the reaction diffusion system, and the quality of simulation rendered. As you can see there are multiple waves being cascaded through the sculpture.

Reaction Diffusion Formula – Gray-Scott Model

$$A' = A + (D_A \nabla^2 A - AB^2 + f(1 - A))\Delta t$$

$$B' = B + (D_B \nabla^2 B + AB^2 - (k/3 + f))\Delta t$$

Variables	*Description*
A'	This variable is the new value of chemical A or B
A	This variable is the current value of A or B
D_A	This variable is the diffusion rate of chemical A or B (Typically should be $D_A = 1$ and $D_B = .5$)
$\nabla^2 A$	This variable represents the Laplacian value for the current particle for chemical A or B (see section 'Laplacian Matrix' for more)
AB^2	These variables represent the rate of reaction between the chemicals (see Reaction and Feed/Kill Rates)
f	This variable represents the 'feed rate' of the chemicals (see Reaction and Feed/Kill Rates)
k	This variable represents the 'kill rate' of the chemicals (see Reaction and Feed/Kill Rates)
Δt	This variable represents a scaling change in time

Sims, Karl, Reaction Diffusion Tutorial
http://www.karlsims.com/rd.html

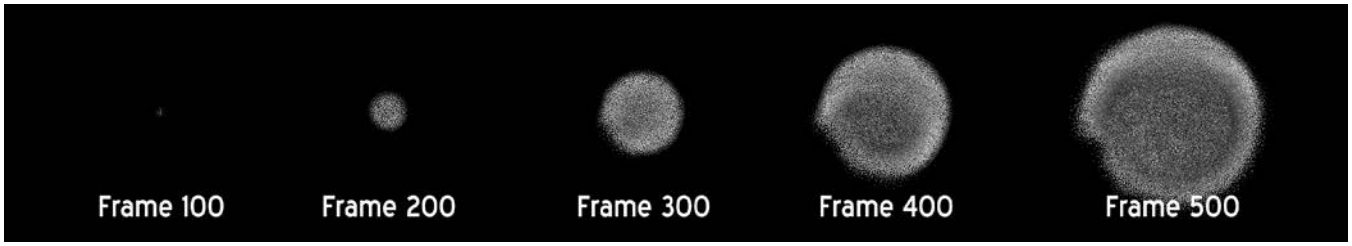
I highly suggest reading Karl Sims' explanation of the Gray Scott model as it explains reaction diffusion extremely well with helpful visuals. The formula above was very slightly changed from the original Gray Scott model which uses a full kill rate rather than (k/3). This change was to compensate for the speed of reaction. The reaction diffusion method is typically used on a two dimensional surface such as a screen in which the model is run over each pixel every frame of the animation. To extrapolate into three dimensions the only serious change to be made was on the calculation of the Laplacian Matrix. The simulation starts with everything flooded with a representational 'Chemical A' and initially seeded in a small area with 'Chemical B'. The chemical values are between 0-1 and are stored in the X and Z positions of vector 'chemical' for easy keeping and calculations. Typically time scale should be left at 1 and not drastically changed.

Laplacian Matrix

The idea of a Laplacian Matrix is to describe a single point's relationship to its neighboring points. Its true mathematical use is much more complicated but in this case it can be easily thought of as a single numeric interpretation of the values of surrounding points.

```
vector laplacian = {0, 0, 0};
int count = ch("../../../../count");
int neighbors[] = nearpoints(geoself(), v@P, 50,count);
int point;

foreach(point; neighbors) {
    laplacian += point(geoself(), "chemical", point);
}
laplacian -= v@chemical*count;
```

This influence of surrounding points allows for the chemicals to artificially diffuse through the surrounding points as if they were naturally spreading around. The code used creates an array of points called 'neighbors' that stores the point values of the nearest X amount of points. Typically the surrounding 10-15 points worked very well and allowed for initial adequate diffusion of 'Chemical B'. These points are then queried for their 'chemical' value which are all added together. This gives the values of all the surrounding points. The current chemical value multiplied by the count of points. This end amount gives the relationship of points around so that if a point is valued at 0 surrounded by 1's that value is easily raised but as that 0 is raised it gets harder and harder to raise it. This when coupled with the Diffusion Rates, this provides accurate diffusion of the 'chemical' values through points.



Reaction and Feed/Kill Rates

The explanation of the reaction is rather straight forward:

- Chemical A is introduced based on the feed rate
- 2 B Chemicals react with 1 A Chemical to convert the A Chemical into a B Chemical
- B Chemical is removed based on the kill rate
- The feed rate is scaled by (1 - A) so that the A Chemical value doesn't increase over 1
- The kill rate is subtracted from the formula to simulate removal and is scaled by B so that the B Chemical value doesn't fall below 0

Kill and Feed Rates are extremely sensitive and yield drastically different behaviors

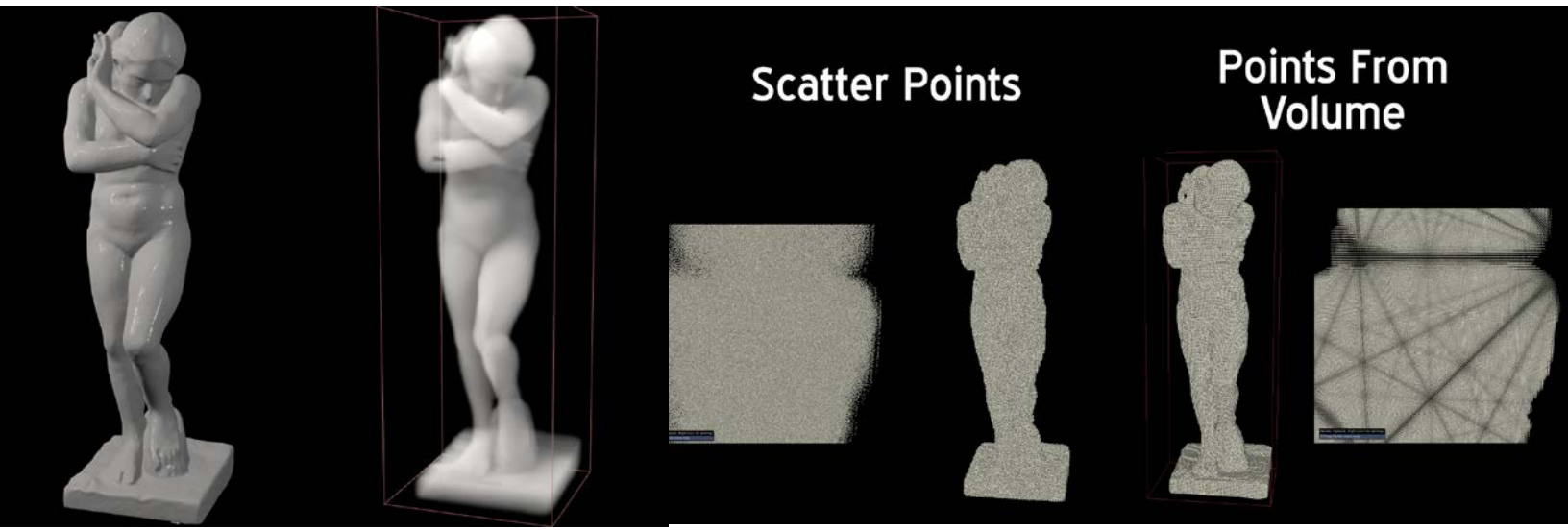
Sims, Karl, Reaction Diffusion Tutorial
http://www.karlsims.com/rd.html

Problems and Workflow

One of the biggest problems for me to start was how I wanted to implement the simulation. I saw two obvious methods, a volume primitive and a point cloud. Initially I believed a volume to be a much more intuitive solution but with various testing and implementations I learned that having the ability to so easily change and manipulate particle attributes was greater than having the quick and straight forward grid that voxels provide. To implement the rest of the formula was not as straight forward as I had hoped. Various values seemed to still grow beyond 1 so various values had to be clamped between 0 and 1 to ensure correct values. Once the new values are calculated they are set again to the chemical vector and then Chemical B (chemical.z) is set into Color Data for visualization in the viewport

```
float diffuseA = ch("../../../../diffuseA");
float diffuseB = ch("../../../../diffuseB");
float kill = ch("../../../../kill");
float feed = ch("../../../../feed");
float timeStep = ch("../../../../time");


float chemicalA = clamp(v@chemical.x, 0, 1);
float chemicalB = clamp(v@chemical.z, 0, 1);

float reactionRate = chemicalA * chemicalB * chemicalB;
float changeA = (diffuseA * clamp(laplacian.x, 0, 1)) - reactionRate + feed *(1 - chemicalA);
float changeB = (diffuseB * clamp(laplacian.z, 0, 1)) + reactionRate - (feed/3+kill) * chemicalB;

float newA = clamp(chemicalA + changeA * timeStep, 0, 1);
float newB = clamp(chemicalB + changeB * timeStep, 0, 1);

@chemical = set(newA, newB, newB);
@Cd = set(@chemical.z, @chemical.z, @chemical.z);
```

After testing on a few primitive cubes I decided a more detailed model was needed for a more serious simulation. I downloaded a scanned sculpture from www.threedscans.com which holds multiple very detailed and high quality models that are quickly downloaded and easily usable. These models make for perfect test models. From this I remeshed the scan to reduce the points by roughly 60% while still retaining the majority of detail needed. This mesh is run through an isoOffset to develop a volume and scatter points in it. Using a completely random point scatter method with the Scatter node versus a perfect grid from the Points From Volume node yielded relatively unrecognizable differences.



After running these points through the simulation, they were piped into another Point Wrangle node and remove and points below a certain Value of Chemical B (chemical.z) to have a range of points that best represents the simulation visually. I learned the hard way that I should not cache particles out after this and instead to cache the particle data out immediately after the simulation. By doing so you then have the option to change this range dynamically rather than having a cache that is locked into the range set initially. These remaining points are then piped directly into the Particle Fluid Surfacer to develop a mesh.

Conclusion

All in all I am extremely happy with the resulting system created. It was an interesting problem to wrap my head around and even more interesting to implement manually in VEX. It was extremely tough to research the mathematics behind this idea and even more tough to pull it into three dimensions. Sim times restricted my ability to experiment with different parameters in the Grey-Scott Model which could have resulted in multiple other interesting designs. The settings are extremely sensitive and having to resim at least a thousand frames to accurately view the reaction simulation makes it a very long and laborious process. Once I found the settings for the waves I decided to stick with them. I also would like to do an even higher quality simulation of 2.5 million particles or more, however this would take extremely long and a ton of space to hold. I would definitely like to try on different models to see exactly how different shapes create different behaviors based on the curvature of the mesh. Creating a full, higher quality render of this sim is the next thing I would like to do, create a more interesting scene to view this rather interesting simulation.