CIS 462/CIS562 Computer Animation (Fall 2016)
Homework Assignment 8

(Particle Systems)

Due: Tuesday, Nov. 29, 2016 (must be submitted before midnight)

Please note the following:

- No late submissions or extensions for this homework.
- This is a programming assignment. You will need to use the *AnimationToolkit* code framework from the previous CIS462/562 IK Player assignment
- Unzip the HW8-AnimationToolkit.rar file or copy the files and folders in HW8-adds+mods.rar file into the corresponding AnimationToolkit folders on your local machine (see the readme.txt file for more details).
- Double click on the file "Demo-ParticleViewer.exe" in the AnimationToolkit\bin directory to see a demo of some of the basic particle system functionality you will need to implement in this assignment. Use the corresponding ParticleViewer project to debug the functions you will need to implement in the APartile.cpp class. Double click on the file "Demo-FireworksViewer.exe" in the AnimationToolkit\bin directory to see a demo of fireworks simulation you will be implementing.
- Commit and push the updated project files to your CIS462-562 GitHub repository.
- When you are finished with this assignment, update your GitHub project files.
- Work within the *AnimationToolkit* code framework provided. Feel free to enhance the AntTweakBar GUI interface if you desire to add more buttons or system variables
- You only need to implement the functions in the *.cpp files marked with "TODO" to complete this assignment.
- **<u>NOTE: THIS IS AN INDIVIDUAL, NOT A GROUP ASSIGNMENT</u>. That means all code written by you for this assignment should be original! Although you are permitted to consult with each other while working on this assignment, code that is substantially the same as that submitted by another student will be considered cheating.**

ANIMATION TOOLKIT – PARTICLE SYTEMS

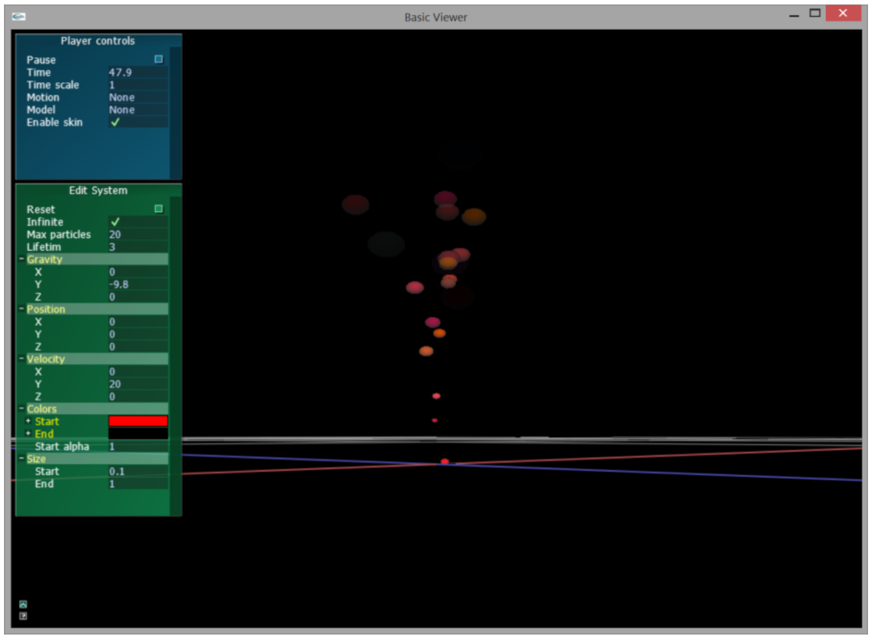
The goal of this assignment is for you to become familiar with the implementation of particle system dynamics and physical simulation. This assignment has three parts:

Part I – Basic C++ Particle System Simulation (*20 points*)

In this part of the assignment you need to implement a basic particle system simulation. The framework provided contains a particle system class whose job is to emit particles. Each particle then has a set of properties -- e.g. position, orientation, color number -- which changes over time.

About the ParticleViewer Project

This assignment builds on the previous assignments, adding a new library and project applications. If you did not complete a previous assignment, you may edit the basecode project files to use the corresponding solution library. To do this, right click on the project, select Properties, and then expand the Linker options. Select 'Input' and then edit the 'Additional Libraries' to point to libAssignmentX-soln.lib instead of libAssignmentX.lib. You will need to do this for both the release and debug versions of your project.



Pause	☐
Time	47.9
Time scale	1
Motion	None
Model	None
Enable skin	✓

Reset	☐
Infinite	✓
Max particles	20
Lifetim	3
- Gravity	
X	0
Y	-9.8
Z	0
- Position	
X	0
Y	0
Z	0
- Velocity	
X	0
Y	20
Z	0
- Colors	
+ Start	🟥
+ End	⬛
Start alpha	1
- Size	
Start	0.1
End	1

The basecode includes a simple interface and 3D viewer which creates a particle system located at (0, 0, 0). There is a UI for setting each particle system option which you can use for testing.

The camera control in the ParticleViewer is the same as in the BVHViewer:

- Left-button drag with the mouse to rotate
- Right-button drag with the mouse to pan
- Middle-button drag with the mouse to zoom
- ‘f’ button will focus the view on the character

Assignment Details

1. (20 points) Implement the various functions in the AParticle.cpp file indicated by “TODO: Insert your code here”

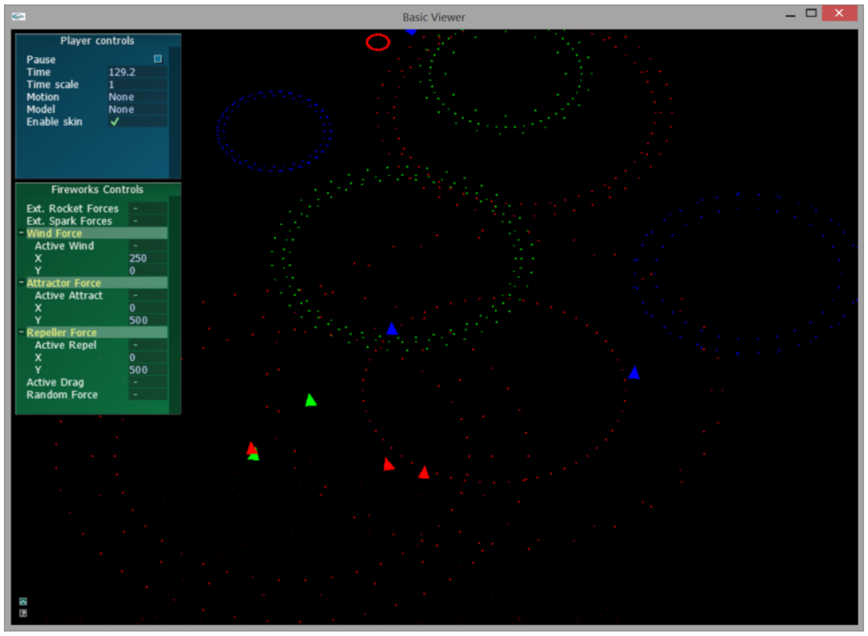
2. (up to 10 points extra credit) Integrate billboards with transparency into the framework to create different effects such as smoke, fire, or butterflies (be creative!)

Part II – C++ based Fireworks Simulation (*55 points*)

In this part of the assignment you will use the particle class implementation developed in part 1 to create a fireworks simulation. When you press the space key, a rocket will fire into the air with a randomized velocity and time to live. When the rocket's time to live expires, it will explode into concentric rings of sparks. When the sparks reach the ground (at Y=0), they should bounce. Both the rocket and the sparks are represented by particles..

About the FireworksViewer Project

The basecode includes a simple 2D viewer for launching rockets. To launch a rocket, press the space key. For the purposes of the framework, particle movement is restricted to the X-Y plane.



Player controls	
Pause	☐
Time	129.2
Time scale	1
Motion	None
Model	None
Enable skin	✓

Fireworks Controls	
Ext. Rocket Forces	-
Ext. Spark Forces	-
- Wind Force	
Active Wind	-
X	250
Y	0
- Attractor Force	
Active Attract	-
X	0
Y	500
- Repeller Force	
Active Repel	-
X	0
Y	500
Active Drag	-
Random Force	-

Assignment Details

1. (55 points) Make some fireworks. Implement the various functions in the ASpark.cpp and AFireworks.cpp files indicated by “TODO: Insert your code here”

2. (15 points) Extra credit.
   - (10 points) Create a rocket trail effect by emitting spark particles based on the rocke position
   - (5 points) Extend the fireworks to 3D

Part III – Houdini- based Fireworks Simulation (*25 points*)

In the third part of this assignment your task will be to create the firework simulation using the Houdini FX software. Houdini is one of the most popular special effects software used in the VFX industry. The main advantage of Houdini is that it is node based and procedural, which means that you can change simulation parameters at any point in the work flow and get desired results instantly.

In this part of the assignment you need to do the following:

1. Download and install the Houdini Apprentice software from http://www.sidefx.com/index.php?option=com_content&task=blogcategory&id=245&Itemid=400 to create the fireworks simulation.
2. Check out the following links for info about Houdini and its user interface:

   Houdini:

   http://tangdonna.blogspot.com/2011/04/sops-dops-pops-rops-vops-shops-and.html

   User Interface:

   http://www.sidefx.com/index.php?option=com_content&task=view&id=1869&Itemid=347
3. Although Houdini has a default option to create a fireworks simulation, in this assignment you will need to create the whole fireworks simulation from scratch using the following nodes and networks since they are the most important components when building a dynamic simulation:

   Nodes: Split nodes, Group nodes, and Collect nodes

   Networks: Geometry network, Particle operator network and Shader operator network.
4. You then need to render the fireworks simulation using the Mantra renderer to create a video demo.

   Extra credit. (*10 pts per effect*). In a manner similar to the C++ fireworks simulation create a spark effect which has particles being emitted when the falling particles collide with the ground. The important nodes to use here are event nodes and collision nodes (which will also help you understand how expressions are used in Houdini). Other effects to be implemented include the creation of different types of "firework patterns" after the rocket explodes through use of wind, attractive forces, repulsive forces, force fields, etc.
5. Please submit your Houdini fireworks project along with the video to the CIS462/562 Canvas site. Before you submit your assignment to Canvas, please put your Houdini project files into a zip file with the name "HW8_LastnameFirstnameHoudini.zip". Also include in this zip file the associated video in either the MOV, MPEG4, or WMV formats (no AVI files PLEASE!) Finally include a description of each of the nodes you used in your simulation and explain the difference between the fireworks simulation with and without the event nodes.

Base Code Details:

Particle System Project Implementation Overview

The class AParticleSystem manages a set of particles relative to a root transform. AParticleSystem also initializes default starting values for each particle. When AParticleSystem creates or emits a particle, it initializes the particle's values using these defaults.

- Particle systems can be infinite, meaning they continuously emit particles (e.g. for smoke), or finite, meaning they only emit a given of particles (e.g. for fireworks). In the next assignment, you will use the particle system created here to create fireworks effects.
- To keep memory needs constant, the particle system has a max number of particles. Particles are "recycled" when one particle dies and it is time to emit a new particle.
- The particle system manages external forces which are applied to its particles each frame. The default framework in the ParticleViewer project supports gravity only.
- The particles in the particle system can be configured to "jitter" values so that particles do not all look identical.

The class AParticle contains a set of properties which can change over time. Some of the important variables include:

- lifespan: how long in seconds a particle will live. For an infinite particle system, a particle is reborn after it dies.
- timeToLive: the amount of time left before the particle dies
- state: a vector (i.e. a 1D array) containing the state of the particle in terms of its position, velocity, forces, mass and time to live.
- stateDot: a vector (i.e. a 1D array) containing derivatives of the state vector
- Pos: the position of the particle
- Vel: the velocity of particle
- Color: the current color of the particle. The color can change over the lifetime of the particle based on the start/end values
- Scale: the current color of the particle. The scale can change over the lifetime of the particle based on the start/end values
- Alpha: the current transparency of the particle. The transparency can change over the lifetime of the particle based on the start/end value
- mass: the mass of the particle

Particle position (**p**) and velocities (**v**) are updated each frame based on the forces applied using either Euler or Runge Kutta2 integration. For example, with Euler integration:

$$\begin{aligned}\mathbf{p}(t_{k+1}) &= \mathbf{p}(t_k) + \Delta t \cdot \mathbf{v}(t_k) \\ \mathbf{v}(t_{k+1}) &= \mathbf{v}(t_k) + \Delta t \cdot \mathbf{a}(t_k)\end{aligned} \qquad \text{where } \mathbf{a}(t_k) = \mathit{force}(t_k)/m$$

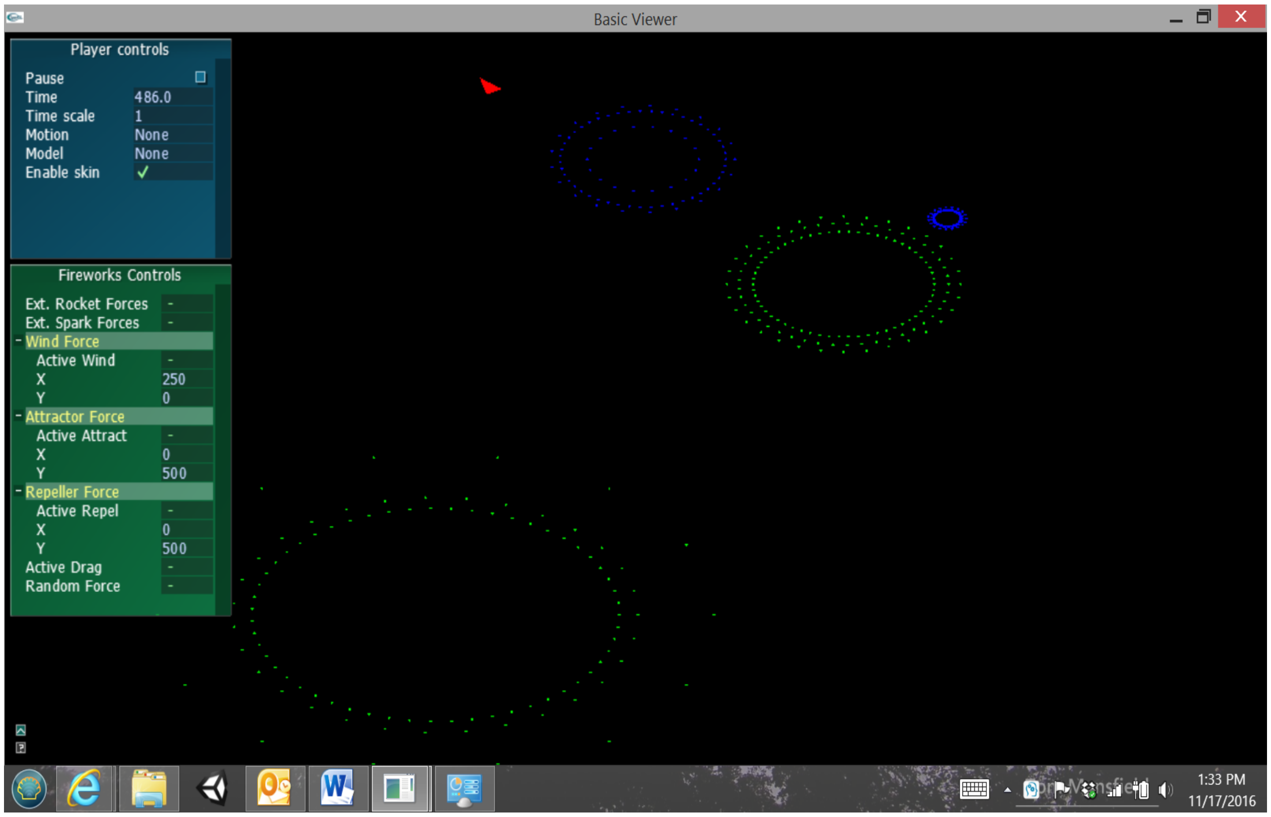
Fireworks Project Implementation Overview

AFireworks simulates rockets and sparks in the simulation through calls to the *AFireworks::update()* function.

At first, there are no particles, so nothing is displayed. When the user hits the space key the function *AFireworks::fireRocket()* function generates a rocket with random velocity (in 2D). As the simulation continues (the user can also fire more rockets). Once launched, the rocket will reach its maximum height (due to gravity) and explode.

From this point in the simulation onward, for *TOTALEXPLOSION* time steps, *AFireworks::explode()* function will generate 5 rings of sparks evenly distributed. The number of sparks per ring and the explosion velocity are random numbers (see comments in *AFireworks* cpp files for details). Sparks are also simulated in the system and have a fixed life time. They might contact the ground, and bounce off with a given coefficient of restitution.

Here's a screen shot of the application:



```
Player controls
Pause
Time            486.0
Time scale      1
Motion          None
Model           None
Enable skin     ✓

Fireworks Controls
Ext. Rocket Forces   -
Ext. Spark Forces    -
- Wind Force
  Active Wind        -
  X                  250
  Y                  0
- Attractor Force
  Active Attract     -
  X                  0
  Y                  500
- Repeller Force
  Active Repel       -
  X                  0
  Y                  500
Active Drag          -
Random Force         -
```

Important Class Details

AParticle is the base class of all particles. Every particle (either a rocket or a spark) inherits from this base class.

Relevant Member variables:

dim: Dimension of the state vector. By default dim = 12;

state: State vector.

- *0 : position x*
- *1 : position y*
- *2 : position z*
- *3 : velocity x*
- *4 : velocity y*
- *5 : velocity z*
- *6 : force x*
- *7 : force y*
- *8 : force z*
- *9 : mass*
- *10 : timeToLive*
- *11 : not defined*

stateDot: Derivative of the state vector.

deltaT: Delta time of one simulation step

alive: Indicates if the particle is dead or not. If it is dead, it should not be displayed and should not be continued to be simulated.

Relevant Member functions:

GetState(): Gets the state vector. Returns the pointer to the state vector

SetState(newState): sets the state vector with value newState. Therefore, newState and state vector must have the same dimension.

Virtual functions are implemented by each sub-class that inherits from the *AParticle* class. Some of the important ones include:

ComputeForce(): Computes the forces to be applied to the particle.

ComputeDynamics(): Compute the derivative of the state vector (i.e. stateDot)

UpdateState(): Update the state vector with the derivative vector and perform other necessary calculations

update(deltaT): Computes one simulation step using either Euler or RK2 integration.

ASpark is a sub-class inherited from the *AParticle* class. It inherits everything (member variables and functions) from *CParticle* and has its own properties.

Relevant Member variables:

color[3]: spark color. It is set by the rocket that it is emitted from.

COR: Coefficient of restitution. Determines how much the spark bounces when it hits the ground.

Member functions you need to implement. See cpp file for details.

computeForces(): Computes the forces acting on the particle. These include gravity plus any other forces such as wind, attraction, repulsion, random, etc.

resolveCollisions(): Computes changes to the velocity vector when the particle hits the ground based on the coefficient of restitution (COR)

ARocket is a sub-class derived from the *ASpark* class. It inherits everything (member variables and functions) from *ASpark* and has its own properties.

Relevant Member variables:

explosionCount: Time to go of explosion phase. It should count down from TOTALEXPLOSION to 0 during a explosion.

mode: Current mode of the rocket, see enum ROCKETMODE {FLYING, EXPLOSION, DEAD}.

Vexplode: min vertical velocity for rocket to explode

AFireworks creates and simulates the rockets and sparks.

Relevant Member variables:

vector<CRocket> rockets*: STL Vector of pointers to *ARocket*. Rockets in this vector can either be flying or generating sparks

vector<CSpark> sparks*: STL Vector of pointers to *CSpark*. Sparks that are generated by the exploding rockets.

rocketMass: mass for rocket (default is 50).

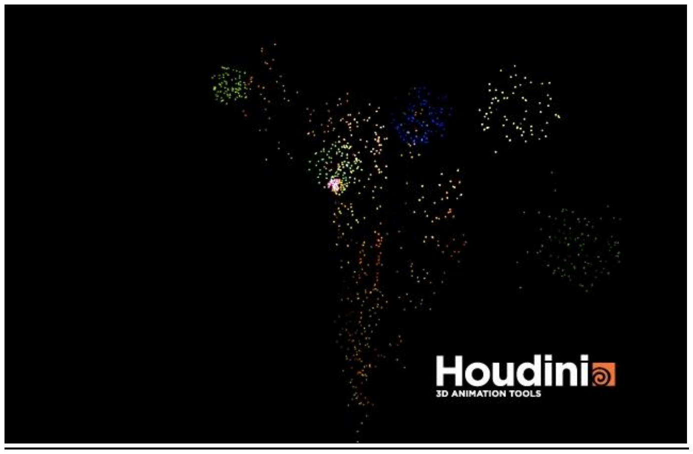
sparkMass: mass for rocket (default is 1).

Functions that you need to implement. cpp file for details.

void explode(): When a rocket reaches its maximum height, it explodes and calls this function to generate sparks

void fireRocket(): When a key is pressed, this function is called to generate a rocket.

Part III – Houdini- based Fireworks Simulation



Step 0: If you haven't done so already, go to http://www.sidefx.com/ and download the free Apprentice version that is available. The Apprentice version will insert a small watermark in your renders, but that is fine for this assignment.

Step 1: Create a Geometry node inside the object network. Enter inside the geometry node and create a POP network inside it. A POP network is a particle network. All the particle operations happen inside this network.

Step 2: Now that you have a particle network, enter inside it and create an emitter. The best emitter is the "location" node; it can emit particles from a point which is exactly what is needed for a fireworks simulation. Change the following parameters in the location node as follows:

- Birth –> Constant birth rate: Change this parameter so that the default number of particles emitted is reduced.
- Birth –> Life Expectancy: Change this parameter so that the particles die after a short time
- Birth –> Life Variance: changing this parameter will make the particles die at random times.
- Attributes ->Velocity : add a positive velocity in the y-direction so that the particles go up
- Attributes ->Velocity Variance: Reduce the variance in x and z, so that there is not much variation when the particles are emitted from the launcher.

Step 3: Now that the particles are rising, you need them to fall after a while. This can be accomplished by adding a force node named "gravity". You will need to reduce the gravity force parameter since Houdini is highly scale dependent and a gravity of -9.8 will not work in this case. Use the scale slider to reduce the value in order to get a proper simulation. In Houdini each block in the grid represents 1 square meter so our simulation is actually very small when compared to real world measurements.

Step 4: Now that you have a decent simulation where particles are flying and dying, the next important steps are to:

- add trails to the particles, and
- spawn additional particles when the particles die - just like real fireworks do!

In order to do this you need to create a group for the particles. Groups are important in Houdini as they allow a group of particles to be reused in a network as and when needed.

First you will create a group for all the dying particles. To do this add a GROUP node after the gravity node named “death group”. Next under group->create->rule, check the enable button. This will enable the rule ($DEAD == 1). This means that whenever the particles die, they will be added to the “death group”. Once you have grouped the particles you can then start on creating trails and also blast particles.

Step 5: The main way to spawn new particles is to create a SPLIT node. Split nodes help to split and create new particles from existing particles. To create the trailing particles add a split node after the group node and name it “trail particles”. Change the following parameters in the split node as follows:

- Split -> Birth -> Birth probability : Experiment with values
- Split -> Birth -> Life expectancy : Experiment with values
- Split -> Birth -> Life Variance: Experiment with values. Note: the particles need to die fairly quickly.

After this step, you will see that the particles are emitted with trails and they follow the main particle until it dies. You can add color to these particles by adding a color node, select Ramp and then vary colors as desired. An expression also can be added to make the color vary over the lifetime of a particle. To do this go to color->Color->Ramp->Lookup and type $LIFE

Step 6: Now that you have particles with trails the next step is to create a set of particles that spawn when the main rocket particles die. To do this create another split node after the group node, but this time the split node will be used to create blast particles. This will be accomplished by using the “death group” which was previously created as the source group for the split node. As a result of this newly added split node, when the particles die, they will emit new particles. Parameters in the split node should be changed as follows:

- Split-> Birth-> Birth Probability: Experiment with different values to see what effect is has on the fireworks simulation
- Split-> Birth-> Life expectancy : Experiment with different values to see what effect is has on the fireworks simulation
- Split-> Birth-> Life variance: Experiment with values. Also select kill original particle (or else it will keep emitting new particles)
- Split-> Attributes-> Initial velocity: Change to add inherited velocity
- Split-> Attributes-> Variance: change to any random values that you like

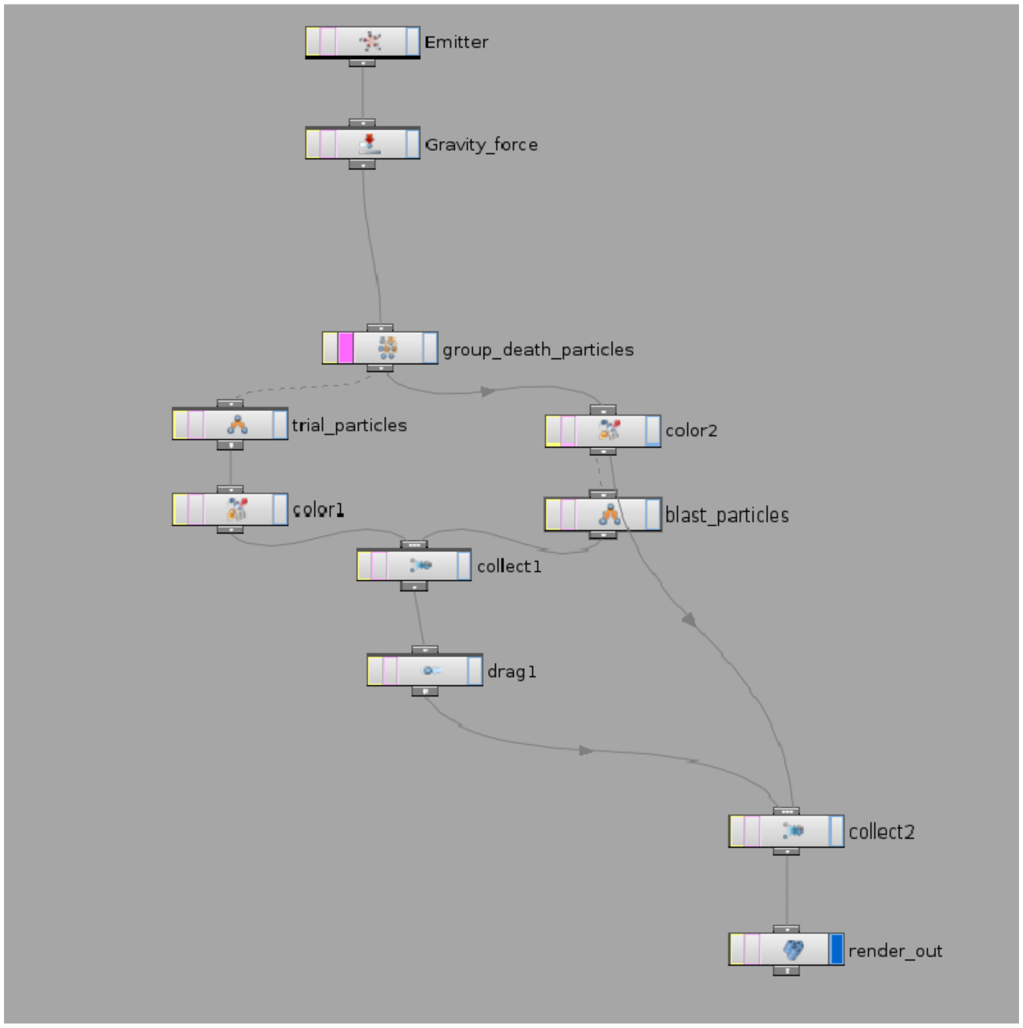
Similar to step 5, you also can add a color node to affect the colors of the newly emitted particles.

Step 7: Now that you have 2 separate streams, it is important to know that once you use a split node, it effectively creates a separate network which means the particles will not carry their properties through the split operation. To remedy this add a collect node and connect the 2 streams to this collect node so that common nodes can be added that will affect both streams.

After the collect node now add a drag node so that you can add a constant drag force to the particles to simulate air resistance.

Finally add another collect node that connects this stream and the color node of the blast particles. Note this step is not necessary if you have the color node after the split node initially.

At this point your Houdini firework simulation should look as follows:



Step 8: The final task in this part of the assignment is to render the simulation. In order to do this:

- first create a light (a point light is fine for this simulation) and a camera. Position and orient the camera to get a good view of the fireworks.
- now add a render particle node after the last collect node. Select the “Render as disks” option for the fire work simulation and also reduce the size of the particles.
- next change from the Geo network to “out” network. Press tab and add the mantra node. Mantra is the default renderer that comes packaged with Houdini. Set the mantra->select render frame range parameter to the desired duration and also specify the folder location on disk where you would like to frames rendered.
- Now hit the render button and the images of your fireworks simulation will be rendered to the disk. Note: you will have to use compositing software, such as Adobe After Effects, to create the fireworks video.