Trail Renderer - Ben Delaney-Brownlow - C00199438

Github URL: https://github.com/BenDB925/RWM3

Introduction

This document describes the trail renderer I am making for RWM3. It is created by instantiating my ParticleManager, which can be created as many times as you want in a scene, and is scalable in terms of performance cost.

Features

Basic Particle System

I will have a particle system that, once instantiated will emit particles in a given direction. The user will be able to change the following aspects of the system:

- Position to parent to. If the user changes this, the system will move along with it.
- The offset from the position to parent to.
- The starting velocity of the particle that are emitted
- The final velocity of the particles
- The variation in velocities
- The time between particle emissions from the system
- The variation of time to live for the particles (so they don't all despawn at the same age)
- The minimum time to live of the particle
- The maximum time to live for the particle
- The size of the particles
- The initial scale of the particle
- The final scale of the particle
- The texture of the particle
- The shape of the emitter

The particles will have a texture that uses alpha blending to fade out gradually.

Primitive Shapes Emitter

This is a modified version of the basic particle system that will emit primitive shapes instead of textures. I will support the following shapes:

- Star
- Triangle
- Square
- Pentagon

The users will be able to modify the following aspects of the primitives:

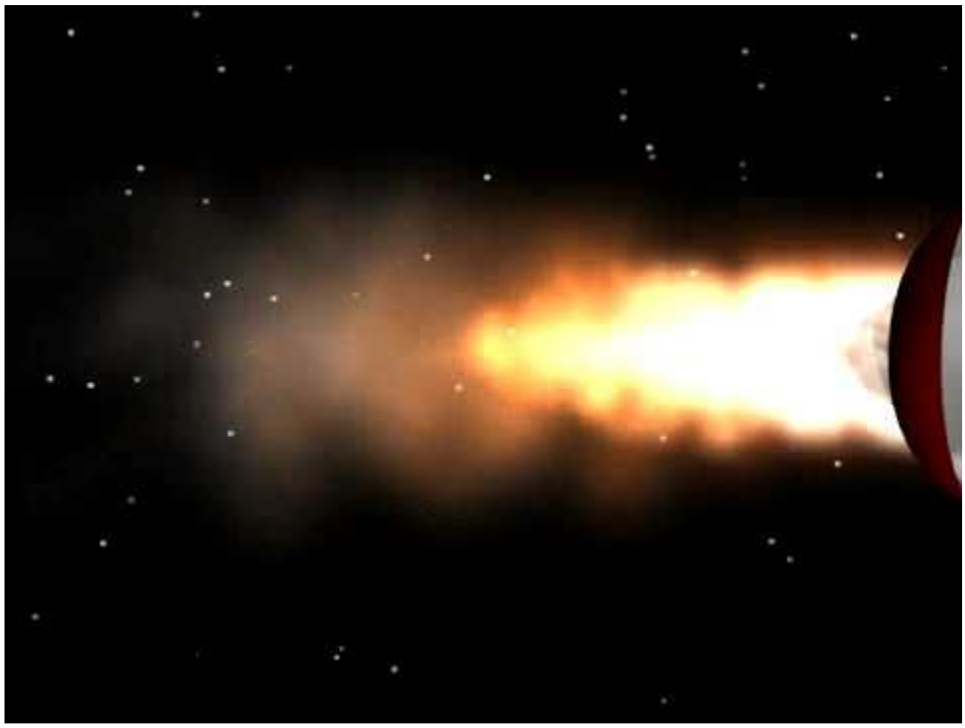
- Size
- Colours it will lerp between throughout it’s lifetime
- Ending Colour
- Alpha Value
- Rotation Velocity

With this system, the user will still be able to use all of the parameters available to the user in the basic particle system - minus the ability to apply a texture to the particles since we are using primitives.

Preset Rocket Thruster Particle System

This will be a fully finished thruster system that will only need a position, or an object to parent itself to, in order to be viewed on screen.

The system should look like the below image once loaded.



Preset Footsteps Particle System

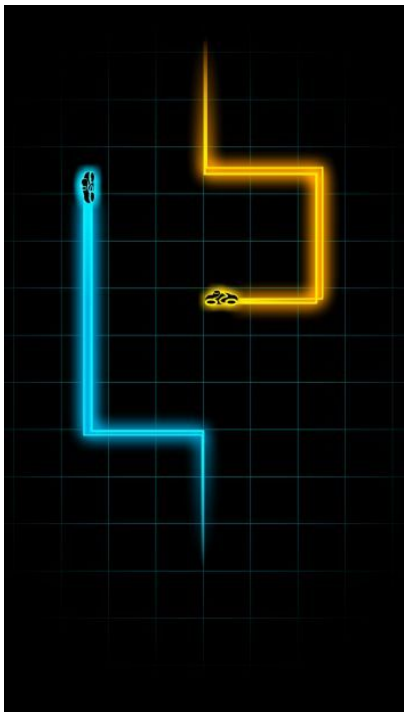
This system, similar to the Rocket Thruster particle system, will only need a position, or an object to parent itself to, in order to be viewed on screen. The textures, colours and scaling will be already set up.

The effect will look similar to this, only with an alpha value so it will blend with the background.

Preset Tron-Style Particle System

This system will be a pre-configured particle system.

The particle system will look like the image below when loaded in.



Preset Star Particle System

This system will be a pre-configured particle system using primitives.

The particle system will look like the image below when loaded in.

How to use this component

How to make custom emitter shapes

There are five different shapes available for the user to make their particles emit from: Point, Rectangle, Ring, Circle and Triangle. Here are the customisations available to each one:

- Point
    - No customisation, the particles will emit from the point specified
- Rectangle
    - The width and height of the rectangle
- Circle
    - The radius of the circle
- Ring
    - The radius of the void in the centre of the circle, and the outer circle radius
- Triangle
    - The user can specify the three vertices of the triangle
    - These vertices must be between 0,0 and 1,1
    - The central point, 0.5,0.5 will be the position of the particleManager, so the triangle will be relative to that

```
#include "ParticleManager.h"
ParticleManager _particleSys;

ParticleManager::ParticleSettings settings = ParticleManager::ParticleSettings();
```

To initialise a point shaped emitter

```
settings._emissionShape = new ParticleManager::EmissionPoint();
```

To initialise a rectangle shaped emitter

```
//the below code will initialise the emitter to be of width 100, and height 50
settings._emissionShape = new ParticleManager::EmissionRect(100,50);
```

To initialise a circle shaped emitter

```
// this will create a circular emitter with radius 50
settings._emissionShape = new ParticleManager::EmissionCircle(50);
```

To initialise a ring shaped emitter

```
// this will create a circular emitter with radius 50
settings._emissionShape = new ParticleManager::EmissionRing(50, 100);
```

To initialise a triangle shaped emitter

```
// this will create a circular emitter with vertices, (0,0), (1,0), (0.5,1), and scale 50
settings._emissionShape = new ParticleManager::EmissionTriangle(
                      Vector2(0, 0),
                      Vector2(1, 0),
                      Vector2(0.5f, 1),
                      50);
```

Example: Making a custom texture trail

In the below example, any fields that I don't initialise explicitly in the settings variable will be set to default.

```
//position of the particle manager
settings._positionToParentTo = _mousePos;
//initial velocity of the particles
settings._startingVelocity = 150;
//final velocity of the particles
settings._endingVelocity = 0;
//the time between particles being emitted
settings._emissionRate = 0.005f;
```

```
//the rotation speed of the particles
settings._rotationSpeed = 10;

//the shape of the emitter
settings._emissionShape = new ParticleManager::EmissionPoint();

//the direction the particles will move in
settings._startingDirection = new Vector2(0, 1);


//the initial scale of the particles
settings._startScale = 0.3f;

//the final scale of the particles
settings._endScale = 3;

//the texture to draw
settings._texture = TextureLoader::loadTexture("assets/particle.png", _renderer);

_particleSys = ParticleManager(settings, _renderer);
```

Default Values:

- _positionToParentTo: new Vector2(300, 300)
- _offsetFromParent: Vector2(0,0)
- _startingVelocity: 0
- _velVariation: 0.25f
- _emissionRate: 0.02f
- _minTTL: the length of the colour transitions supplied, if no transitions are supplied, it is the length of the default transition, which lasts for 4 seconds
- _maxTTL: the length of the colour transitions supplied, if no transitions are supplied, it is the length of the default transition, which lasts for 4 seconds
- _particleSize: 1.6f
- _texture: nullptr
- _startScale 1
- _endScale 1
- _emissionShape new EmissionPoint()

Example: Making a custom primitive trail

In the below example, any fields that I don't initialise explicitly in the settings variable will be set to default.

```
ParticleManager::ParticleSettings settings = ParticleManager::ParticleSettings();
//position of the particle manager
settings._positionToParentTo = _mousePos;
//initial velocity of the particles
settings._startingVelocity = 150;
//final velocity of the particles
settings._endingVelocity = 0;
```

```
//the time between particles being emitted
settings._emissionRate = 0.005f;

//the type of shape to draw
settings._shapeType =  Shape::ShapeType::Pentagon;

//the rotation speed of the particle
settings._rotationSpeed = 0.05f;

//the shape of the emitter
settings._emissionShape = new ParticleManager::EmissionCircle(50);

//the direction the particles will move in
settings._startingDirection = new Vector2(0, 1);

//the initial scale of the particles
settings._startScale = 0.3f;

//the final scale of the particles
settings._endScale = 3;


_particleSys = ParticleManager(settings, _renderer);
```

Default Values:

- _shapeType: Pentagon
- _coloursToLerp: White to black fade out over 4 seconds
- _rotationSpeed: 0

Shared variables with ParticleManager have the same default values.

Example: Making a preset trail

This will initialise the particle system to the default rocket trail I design.

```
#include "ParticleManager.h"
ParticleManager _particleSys;

//this will load in all the presets, this is mandatory to call before using any preset
ParticleManager::LoadPresets(_renderer);

ParticleManager::ParticleSettings settings = ParticleManager::_ROCKET_THRUSTER_PRESET;

//position of the particle manager (not mandatory to overwrite, it will just have a default position otherwise)
settings._positionToParentTo = _mousePos;

//the user can edit any variables to the settings variable here, like in the custom system above, or leave it as the preset.
_particleSys = ParticleManager(settings, _renderer);
```

Example: Updating/Rendering a trail

This will update a created particle system. This will change it's position, create, update and reset particles in the effect etc.

```
_particleSys.update(deltaTime);
```

If a particle system is parented to an object (it's position is the address of another object's position) then the system will automatically follow the other object.

```
SDL_RenderClear(m_p_Renderer);
_particleSys.render();
SDL_RenderPresent(m_p_Renderer);
```

The particle system is editable during the game loop. To do this, you can change the variables like you did at initialisation, only this time you are editing the particle system directly, instead of a settings object. For example, if I wanted to change the particle system's final scale of the particles during the game to 10, I would replace the code block above to read:

```
_particleSys._endScale = 10;
_particleSys.update(deltaTime);
SDL_RenderClear(m_p_Renderer);
_particleSys.render();
SDL_RenderPresent(m_p_Renderer);
```

Demonstration App

Description

The users will see a particle system that will follow the mouse cursor as they move it around the window. The users are able to switch the particle system with any of the presets I listed above, and then edit the variables that make up that system in real time.

Controls

Pressing space will pause the particle system from emitting new particles if it's active now, if it's currently paused, it will resume emission.

The app will feature a small menu along the top of the screen. This is used to edit the current particle system in real time. Users can switch the currently selected item by scrolling along the list using the left and right arrows.
Pressing up or down will increment and decrement the value respectively.

Items on the menu

- The first item on the menu is the type of particle that's emitted, users can press the up and down arrow keys to switch between texture, star, triangle, square, and pentagon, along with the four presets: Rocket thruster, Star preset, footprints and tron. If the user reaches the end of the list of particle types, the selection wraps around to the first item.

- Size of the particle.
    - Pressing up or down will increment the item's value by 0.02 units
    - If the value goes below 0.02, the value is switched to 0.02
- Emission rate
    - Pressing up or down will change this value in increments of 0.005 units
    - If this value goes below 0.005, it's value is changed to 0.005
- Minimum Time to Live
    - Pressing up or down will change this value in increments of 0.2 units
    - If this value goes below 0, it's value is changed to 0
- Maximum Time to Live
    - Pressing up or down will change this value in increments of 0.2 units
    - If this value goes below 0, it's value is changed to 0
- Initial Velocity of the particles
    - Pressing down or up will change the velocity in increments of 5 units
- Final Velocity of the particles
    - Pressing down or up will change the velocity in increments of 5 units
- Variation in the initial velocity
    - Pressing down or up will change the velocity in increments of 5 units
- The angle of the cone of particles that’s emitted
    - Pressing down or up will change the velocity in increments of 0.1 units
    - If the angle is less than 0, it’s value is changed to 0
- If the users have selected a shape based particle system, they will be able to edit the colour of the shape that's emitted with 4 items, representing the R G B and A values
    - Pressing the up or down arrows will change the item's value in increments of 5
    - If the value goes below 0, the value is switched to 255
    - If the value goes above 255, the value is switched to 0
- There is another 4 items for the second colour of the particle which follows the same format as above

At the bottom of the screen is the user’s currently selected emitter shape. The user can use the ‘A’ and ‘D’ keys to cycle through the different preset shapes I have made for the purpose of the demo.

- If the user scrolls through all available shapes using the ‘A’ or ‘D’ keys, the list of preset shapes will wrap around again to the first shape.