Particle System

Sergi Colomer Ferrer

Index

1- Introduction

- What is a particle system?
- Organic effects
- About my program

2- Market Study

- Sprite sheets
- 3D

2.1- Components

2.1.1- Particle System

- Definition
- Wiliam T. Reeves
- Hierarchy

“A particle system is a collection of many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system.”

2.1.2- Emitter

- Definition
- List

2.1.3- Particle

- Definition
- About atlas
- David Finseth

“Particle effects are a unique tool that can add interactivity and responsiveness to your games. They excel at creating a lot of movement and impact. Particle effects can be used to create magical fireballs, swirling dimensional portals, or for directing the player’s attention to a glowing treasure chest.”

3- My Approach

3.0- Features

- 2D implementation
- Particles movement will be linear but will have the option to interpolate between the initial and final speeds
- A particle atlas
- The data properties will be compacted in a single function called “LoadEmitterData()”
- Use of pools/lists
- Different types of emitters
- Emitter properties

3.1- Particle System Module

In charge of:

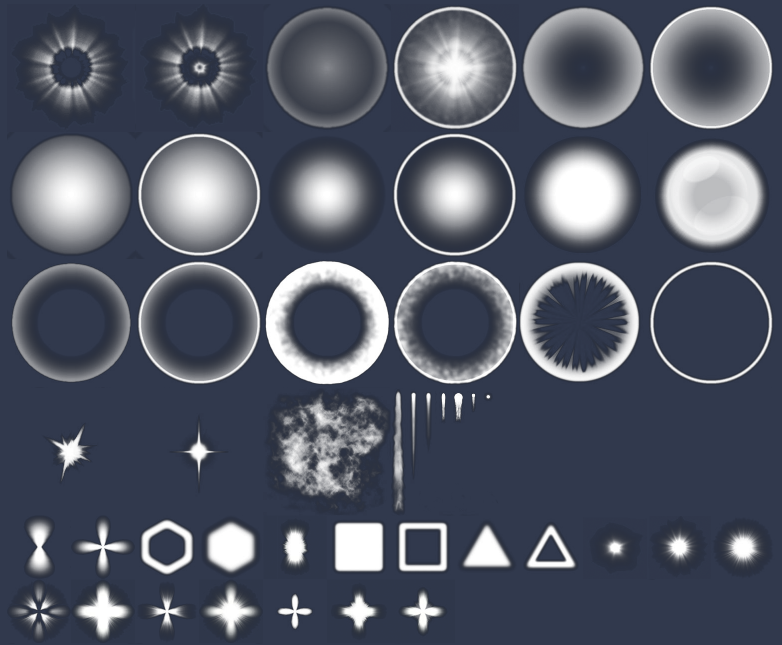
- Spawning / Despawning emitters
- Updating emitters
- Emitters destruction process
- Particle atlas path
- Load emitters data

3.2- Emitter Class

In charge of:

- Particle parameters
- Spawning / Despawning particles
- Updating & PostUpdating (Drawing) particles
- Self-despawn process
- Moving the emitter
- Set & Get functions to the vortex activation bool

3.3- Particle Class & EmitterData



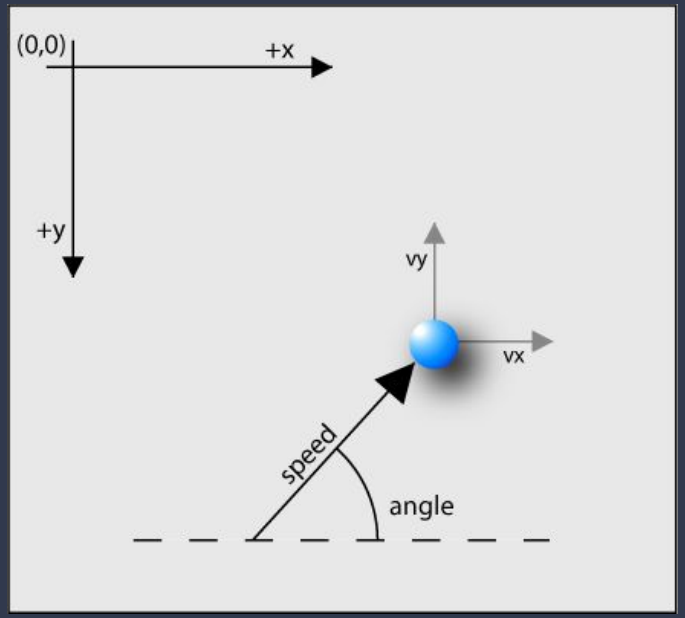
3.3.1- EmitterData

Parameter	Description
rotSpeed	The rotational speed at which the particle will spin.
angleRange	The angle representing the direction the particle will take when moving.
initialSpeed / finalSpeed	The initial and final speed values determining how fast the particle will move.
initialSize / finalSize	The initial and final particle size in the case you want to vary the size of the particle as time goes.
emitNumber	Determines how many particles are spawned each frame.
emitNumMult	An offset to the number of particles spawned each frame.
maxParticleLife	The amount of time a particle spawned by this emitter will last before it is destroyed.
emitLifetime	The maximum amount of time an emitter can be alive regardless if there are particles still alive or not.
texRect	The texture section we want to draw on screen.
initialColor / finalColor	The initial and final color values determining what color the particle will have at every frame.

Parameter	Description
blendMode	The type of draw we want to use for the particle.
randRotSpeed	Parameter to randomize the rotation speed within the given range.
randInitialSpeed / randFinalSpeed	Parameters to randomize the initial and final speeds within the given ranges.
randEmitMult	Parameter to randomize the offset of the particle spawning within the given range (so it's even more random).
randLife	Parameter to randomize the life of the particle within the given range.
randInitialSize / randFinalSize	Parameters to randomize the initial and final particle sizes within the given range.
vortexActive	Parameter to determine if the particle should be affected by the vortex or not.
halfTex	Parameter to determine if the particle uses only half the texture.
eType	The type of emitter. To differentiate an emitter from another one.

3.3.2- Particle Class

- In essence



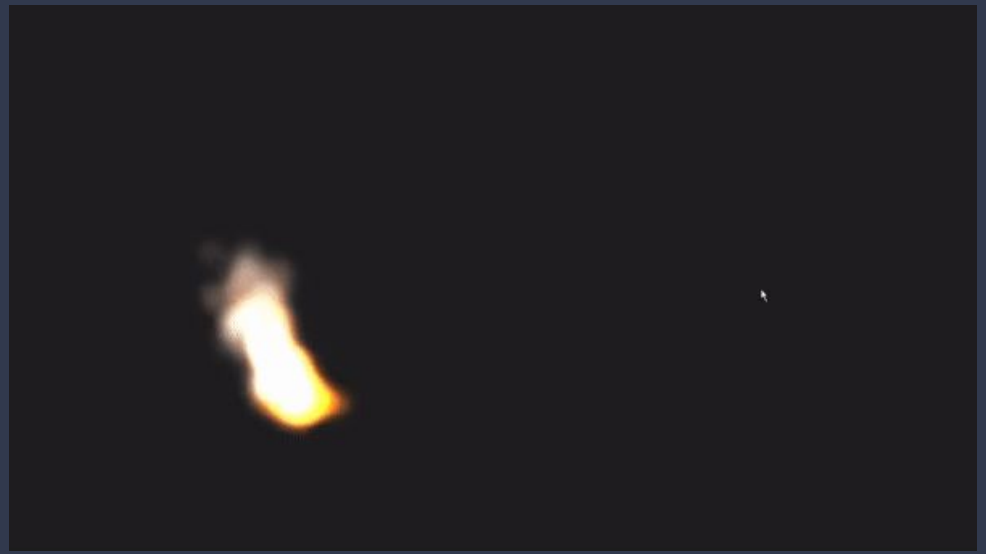
3.3.2- Particle Class

```
class Particle
{
public:
        // Constructor
        Particle(variables used to create the particle);

        // Called each loop iteration
        void Update(float dt);

        // Draws the particle
        bool Draw();

private:

        fPoint pos;
        fPoint curSpeed;
        fPoint curSize;

        SDL_Rect pRect;

        uint life;
  };
```

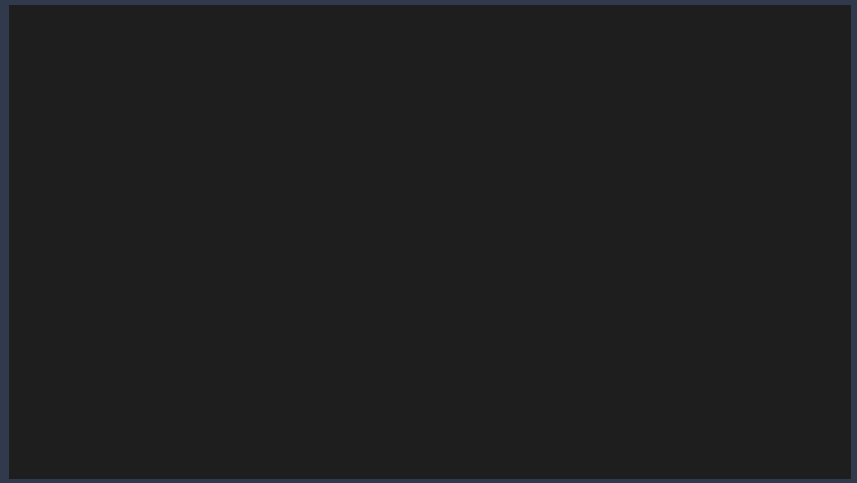
```
curSpeed.x = speed * cos(DEG_2_RAD(angle));
curSpeed.y = -speed * sin(DEG_2_RAD(angle));

pos.x += curSpeed.x * dt;
pos.y += curSpeed.y * dt;

// Particle's life decrease
life--;
```

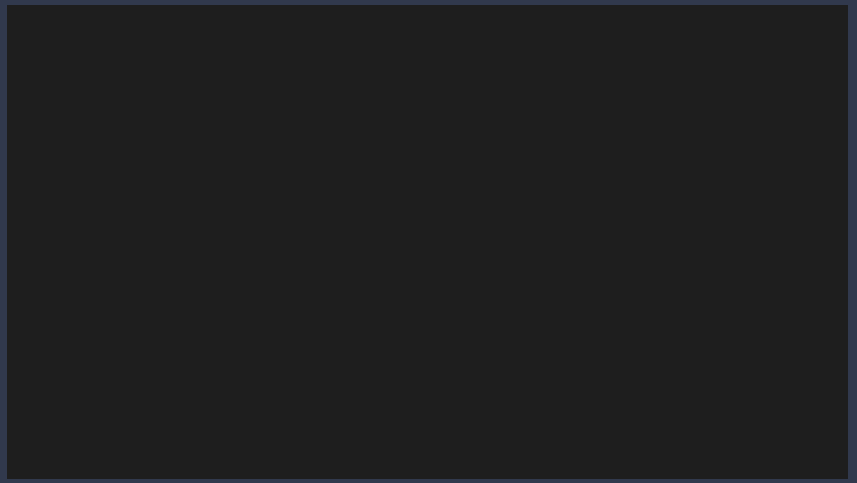
3.4- Special Functions

3.4.1- Moving fires

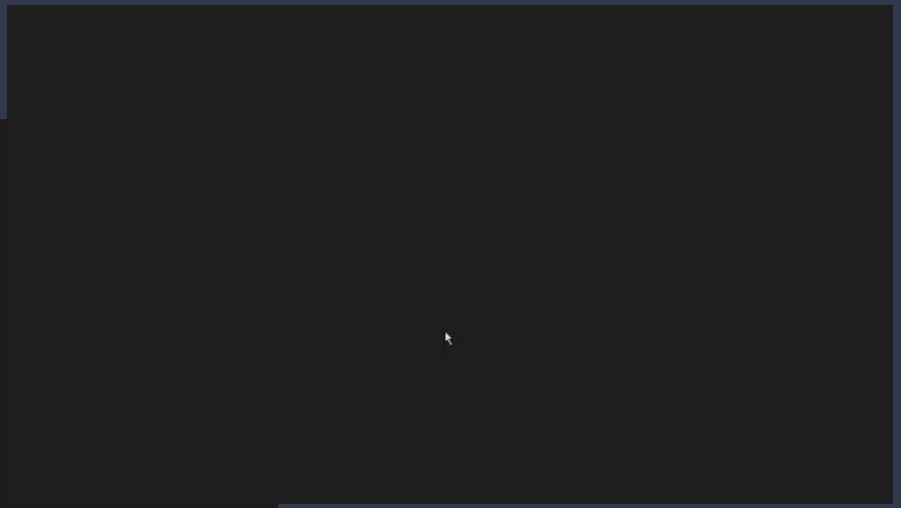


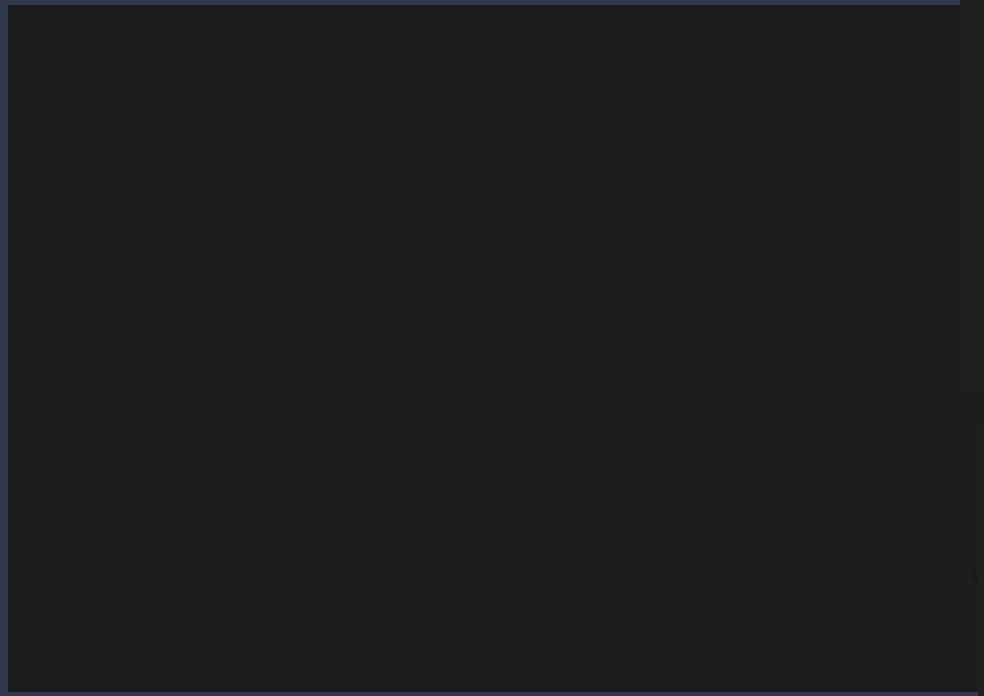
3.4.2- Vortex

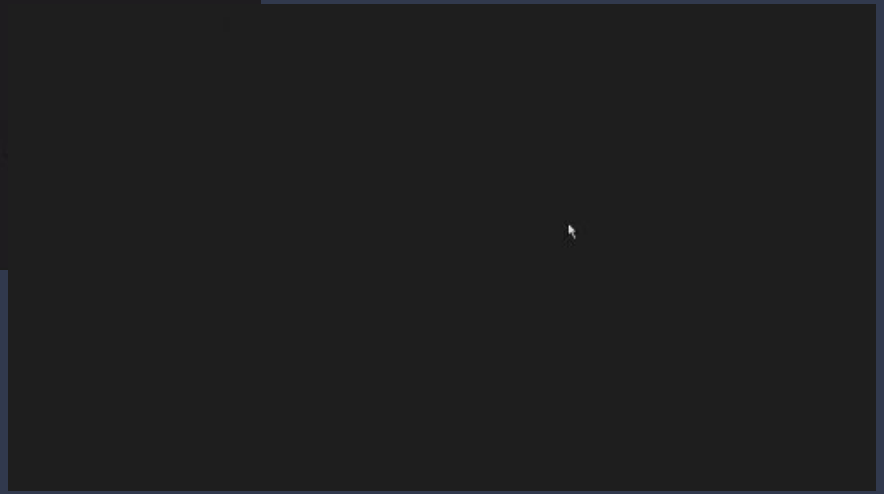


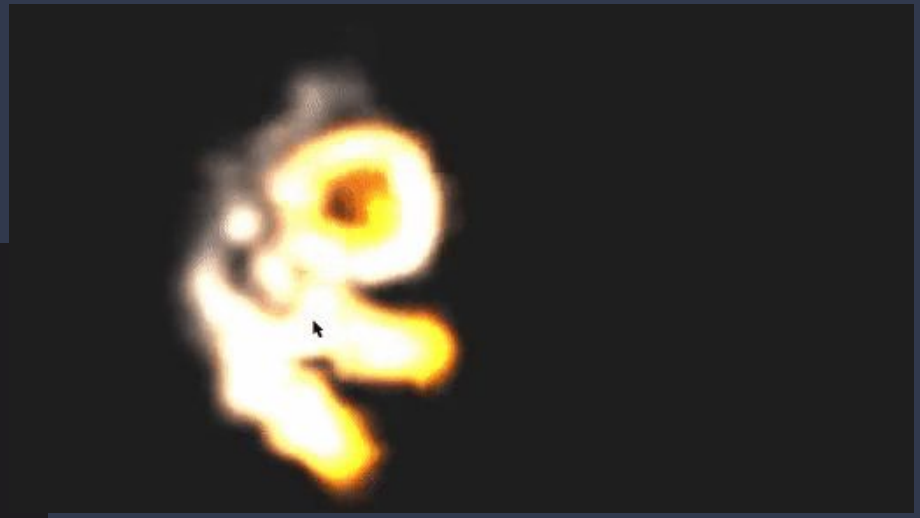


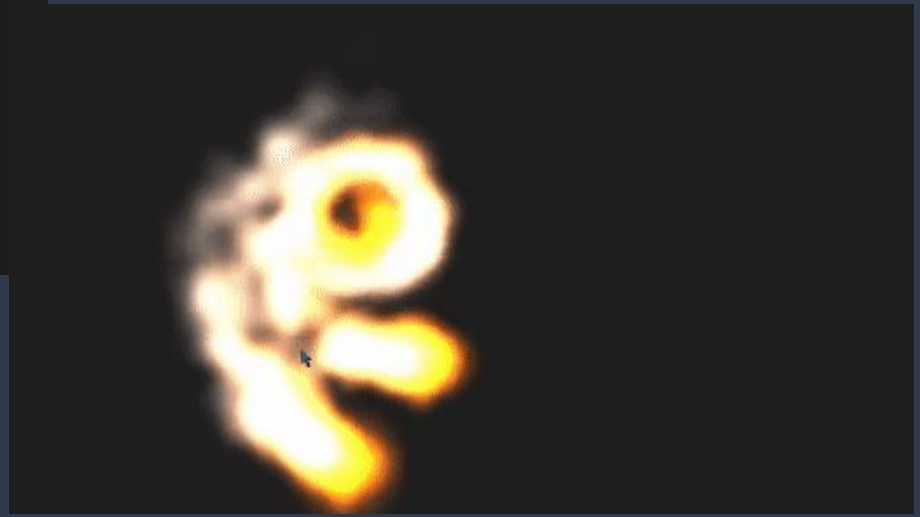
3.5- End Results











4- Possible Improvements

- Vortex Mechanic
- Subemitters
- Multiple phase emitter
- Animated particles
- Physics
- Collisions
- Half texture

5- TODOs & Solutions

TODO 1: Set up the emitter

```cpp
Emitter::Emitter(fPoint pos, EmitterData data)
{
    srand(time(NULL));

    active = true;
    /* TODO 1: Setup the emitter, for that you need to:
        - Store the given variables
        - Calculate the maximum particles that can spawn each frame
        - Start the "lifeTimer" unless you want the emitter to be permanently active
    */

    // Clear the particle list
    ListItem<Particle*>* p = particlePool.start;
    while (p != nullptr)
    {
        ListItem<Particle*>* pNext = p->next;
        DestroyParticle(p->data);
        p = pNext;
    }
    particlePool.Clear();
}
```

TODO 2: Particle constructor

```
Particle::Particle(fPoint pos, float initialSpeed, float finalSpeed, float angle, double rotSpeed, fPoint initialSize, fPoint finalSize, uint life, SDL_
{
    /* TODO 2: Create the particle using the given variables
        - Change the speeds regarding the angle
        - Don't forget to reset "timeStep"
    */
}
```

TODO 3.1: Updating alive particles

```
    // Emission timing calculations
    if (data.emitLifetime > 0.0f)
    {
        if (lifeTimer.Read() >= data.emitLifetime)
        {
            active = false;
            data.emitLifetime = 0.0f;
        }
    }

    // TODO 3.1: Update all alive particles

    return true;
}
```

TODO 3.2: Drawing alive particles

```cpp
bool Emitter::PostUpdate()
{
    bool ret = true;

    // TODO 3.2: Draw all alive particles
    // TODO 4.2: In the same loop you can delete all the dead particles

    if (particlePool.Count() == 0)
    {
        app->particleSystem->RemoveEmitter(this);
    }

    return ret;
}
```

TODO 4.1: Particle interpolation

```
    /* TODO 5.3: Draw the particle hitbox if in debug mode
        - You can use the same color of the particle for the rectangle
        - Remember to override the alpha value so that you can still see the particle!
    * */

    /* TODO 4.1: Particle interpolation and life check
        - Increase the current rotation of the particle according to the rotation speed
        - Increment the timeStep each frame from the initial life to interpolate colors and size
        - Check if the life surpasses 1.0f and ready it for deletion
    * */

    return ret;
}
```

TODO 4.2: Particle deletion

```cpp
bool Emitter::PostUpdate()
{
    bool ret = true;

    // TODO 3.2: Draw all alive particles
    // TODO 4.2: In the same loop you can delete all the dead particles

    if (particlePool.Count() == 0)
    {
        app->particleSystem->RemoveEmitter(this);
    }

    return ret;
}
```

TODO 5.1: Color interpolation

```
/* TODO 5.1: Interpolate color
   - Using both the initial and the final SDL_Color variables create a function that interpolates them so that the particle transitions from the initial to the final
   - You can use the InterpolateBetween function as a reference
*/

// Drawing particle on the screen
ret = app->render->DrawParticle(app->particleSystem->GetParticleAtlas(), (int)center.x, (int)center.y, &pRect, &rectSize, 1.0f, curRotSpeed);
```

TODO 5.2: Modify *“DrawParticle”*

```
rect.w *= scale;
rect.h *= scale;

// TODO 5.2: Modify the "DrawParticle" function so that the particle can change color, alpha and even blend mode

if (SDL_RenderCopyEx(renderer, texture, section, &rect, angle, NULL, SDL_FLIP_NONE) != 0)
{
    LOG("Cannot blit to screen. SDL_RenderCopy error: %s", SDL_GetError());
    return false;
}

return true;
```

TODO 5.3: Debug mode

```
    /* TODO 5.3: Draw the particle hitbox if in debug mode
        - You can use the same color of the particle for the rectangle
        - Remember to override the alpha value so that you can still see the particle!
    * */

    /* TODO 4.1: Particle interpolation and life check
        - Increase the current rotation of the particle according to the rotation speed
        - Increment the timeStep each frame from the initial life to interpolate colors and size
        - Check if the life surpasses 1.0f and ready it for deletion
    * */

    return ret;
}
```

TODO 6: Make fire

```
void ParticleSystem::LoadEmittersData()
{
    /* TODO 6: Modify the fire values so that they suit the fire you want to make!
        - The one you have here is one I made and really like how it looks (unless you move it a lot)
    */
    EmitterData::EmitterType type = EmitterData::EmitterType::NONE;
    for (int i = 0; i <= MAX_NUM_EMITTERS_TYPE; i++)
    {
        switch (type)
        {
```

TODO 7: Bubbles

```
if (app->input->GetKey(SDL_SCANCODE_F1) == KEY_DOWN)
{
    LOG("Fire emitter init");
    fires.Add(app->particleSystem->AddEmitter(pos, EmitterData::EmitterType::FIRE));
}

// TODO 7(Optional): Try making bubbles as a new Emitter with what you have learned
```

Thank you for your attention