OpenGL

Shader programing

Authors:

- Nicolas Blin
- Ilan Guenet

Models rendering

- Buffer of **vertices** (vec3*)
- Buffer of **normals** (vec3*)
- Buffer of **texture coordinates** (vec2*)

Camera

- Frutsum
- Lookat
- position (vec3)
- target (vec3)

Lights

```
// Fragment shader
[...]

void main() {
    output_color = vec4(0.f, 0.f, 0.f, 1.f);
    for (int i = 0; i < nb_lights; i++)
    {
        float len_to_light = length(light_dir[i]);
        float distance_factor = clamp(1 / len_to_light * strength_light[i], 0.f, 1.f);
        vec3 computed_color = dot(onormal, normalize(light_dir[i])) * texture(texture_sampler, otex_coord).rgb * distance_factor;
        output_color.rgb += clamp(computed_color, 0.f, 1.f);
        output_color = clamp(output_color, 0.f, 1.f);
    }
}
```

- Diffuse color
- Light attenuation
- Sum up the color computed from every lights

Texture

```
vec3 computed_color = dot(onormal, normalize(light_dir[i])) * texture(texture_sampler, otex_coord).rgb * distance_factor;
```

- For every vertex, 2D coordinates in the texture
- Use a openGL sampler2D
- Get the color with `texture(texture_sampler, otex_coord)`

Particles

- A particle is represented as a cube
- Scale the cubes to be small
- Spawn many particles
- Kill the particles that have lived long to respawn new ones (**keep the cycle**)

Update particles

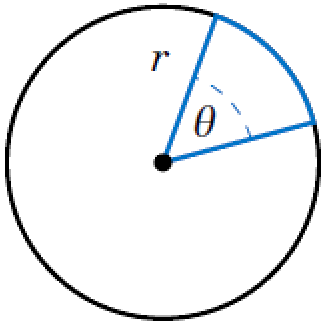
- Update particles' positions
- Update particles' life
- Update particles' color

```
for (Particle& particle : particles_)
    particle.life -= life_diminution;
    if (particle.is_alive())
        particle.position += particle.direction * particle.speed
        particle.color -=  color_attenuation_
```

Generator

```
// Find dead particles
// Respawn the given number of particles
for (Particle& particle : particles_)
    if (particle.is_dead())
        respawn(particle)
        respawned_particle++
        if (respawned_particle == nb_new_particles_)
            break;
```

Gaussian fire



- Gaussian distribution for the radius
- Uniform distribution for the angle
- Uniform distribution to add noises to the particles

```
random_r = normal_distribution();
random_teta = (rand() % 100) / 100.f * 2 * M_PI

random_x = random_r * cos(random_teta)
random_y = random_r * sin(random_teta)

particle.position = (random_x, random_y)
```

Gaussian fire

Updates

- Position is updated up on the top/bottom axis
- Color is updated from yellow (at their spawn) to red (at their death)

Teleporter

- Particles distributed over the circle radius and not randomly inside

```
void teleport()
  if (inside_zone())
    if (entering_zone < threshold)
      ++entering_zone
      portal_generator_A.activate()
    else if (entering_zone == threshold)
      camera.origin_ = portal_generator_B.get_position()
      ++entering_zone

void activate()
  speed_ += 0.005f;
  color_ += 0.005f;
```