Billboarding and 3D Particle System

Advanced Graphics Programming

MEDIA DESIGN SCHOOL GAME DEV

Objectives

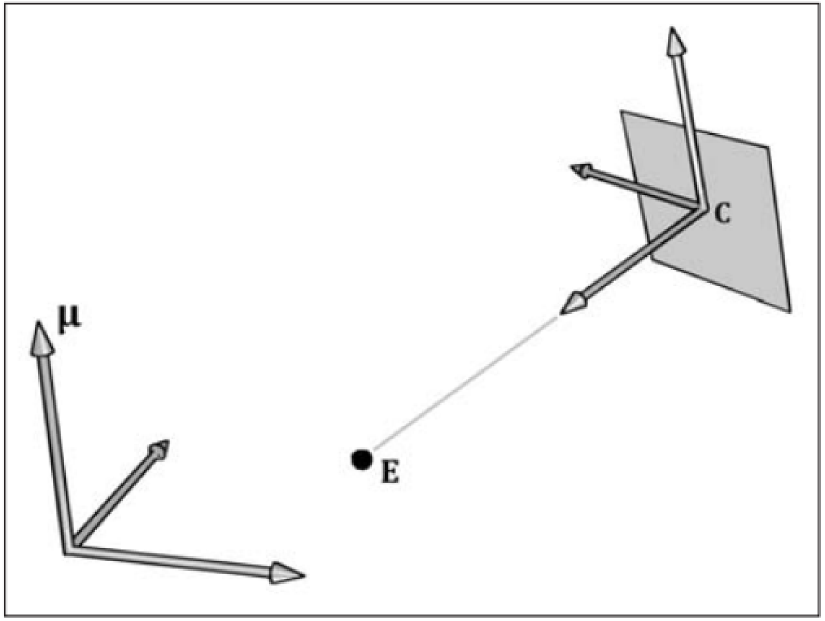
- Create a particle class
- Create a particle system
- Render particles using geometry shader
- Use bill boarding to make particles look at the camera.

Particle

- A particle is a very small object that is usually modeled as a point mathematically.
- It follows then that a point primitive (GL_POINTS) would be a good candidate to display particles.
- However, a point primitive is rasterized as a single pixel.
- This does not give us much flexibility, as we would like to have particles of various sizes and even map entire textures onto these particle

Billboarding Particles

- We will store the particles using points, but then expand them into quads that face the camera in the geometry shader.



Randomness

- In a particle system, we want the particles to behave similarly, but not exactly the same.
- We want to add some randomness to the system.
- For example, if we are modeling raindrops, we do not want all the raindrops to fall in exactly the same way; we want them to fall from different positions, at slightly different angles, and at slightly different speeds.

Randomness

- We create a function to create random values

```
#include <random>

static float randomFloat() {
float r = (float)rand() / (double)RAND_MAX;
return r;
}
```

Particle Class

- The particle class keeps track of all the particles in the system.
- It will have properties like position, velocity, elapsed time, speed.
- We need to track elapsed time as each particle has a lifetime. After the particles lifetime we can either delete the particle or reset its position.
- In the constructor it would need the initial position and all calculations are done in world space.

Particle Class

- The Particle class will need an update function which updates the position of each particle.
- Reduce the elapsed time since the birth of the particle.
- And if the elapsed time is less than zero then we reset the position of the particle.

Particle Class

```
void update(float dt) {

this->velocity.y += -0.2 * .0167f;
this->position += velocity;
this->elapsedTime -= .000167;

if (this->elapsedTime <= 0.0f) {

        this->position = this->origin;
        this->velocity =
        glm::vec3(0.25 * cos(this->id * .0167) + 0.25f * randomFloat() - 0.125f,
                1.5f + 0.25f * randomFloat() - 0.125f,
                0.25 * sin(this->id* .0167) + 0.25f * randomFloat() - 0.125f);
         this->elapsedTime = randomFloat() + 0.125;
 }
}
```

Particle System Class

- The particle system class creates and manages the individual particle and also draws each particle.
- The particle positions are stored in an array after updating each particle.
- Even though the particles are in world space we still have to get the view and projection matrix. So we have to pass in the camera class.
- We need a particle vector class to store all the particles so that it is easy to cycle through all the particles.
- And position vector to store the positions

Particle System.h

```
class ParticleSystem {

public:
     ParticleSystem(glm::vec3 origin, Camera* _camera, std::string texFileName);
     ~ParticleSystem();

     void render(float dt);

     std::vector<Particle> particles;
     std::vector<glm::vec3> vPosition;

private:
    Camera* camera;
    GLuint vao, vbo, texture, program;
    float nParticles;
};
```

Particle System.cpp (Constructor)

- Load texture
- Create program with vs, fs and gs stages
- Init particles

[redacted]

```
nParticles = 4000;
for (int i = 0; i < nParticles; i++) {
vPosition.push_back(glm::vec3(0.0)); //initialize position vector
Particle p = Particle(
    origin, // pos
    glm::vec3(0.25 * cos(i * .0167) + 0.25f * randomFloat() - 0.125f, // vel
                 2.0f + 0.25f * randomFloat() - 0.125f,
                 0.25 * sin(i* .0167) + 0.25f * randomFloat() - 0.125f),
     randomFloat() + 0.125, // elapsed time
     1.0f, // speed
      i, // id
  _camera);   particles.push_back(p); // add
}
```

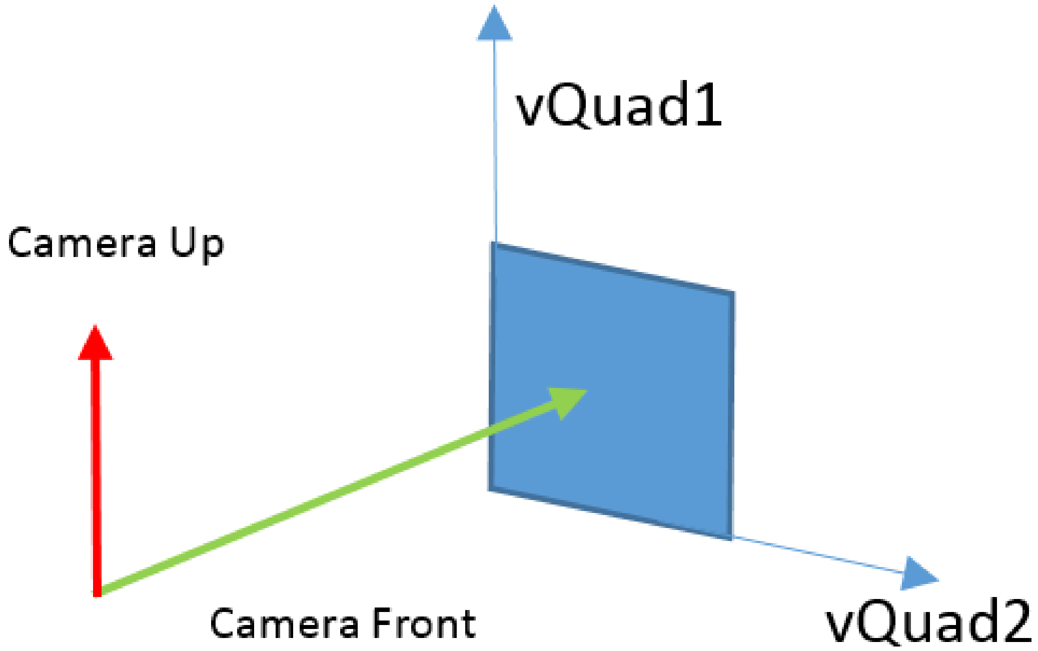
Particle System.cpp (Constructor)

- Set vao and vbo
- Set vertex attributes- only position

```
  glGenVertexArrays(1, &vao);
   glBindVertexArray(vao);

   glGenVertexArrays(1, &vbo);
   glBindBuffer(GL_ARRAY_BUFFER, vbo);
   glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * vPosition.size(), &vPosition[0],
GL_STATIC_DRAW);

   glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(glm::vec3), (GLvoid*)0);
   glEnableVertexAttribArray(0);

glBindBuffer(GL_ARRAY_BUFFER, 0);
   glBindVertexArray(0);
```

Particle System.cpp (Render)

```
for (int i = 0; i < nParticles; i++) {
		particles[i].update(.0167);
		vPosition[i] = particles[i].getPosition();
}
glm::mat4 viewMat = camera->getViewMatrix();
glm::vec3 vQuad1, vQuad2;

glm::vec3 vView = camera->getCameraFront();
vView = glm::normalize(vView);

vQuad1 = glm::cross(vView, camera->getCameraUp());
vQuad1 = glm::normalize(vQuad1);


vQuad2 = glm::cross(vView, vQuad1);
vQuad2 = glm::normalize(vQuad2);
```


vQuad1
Camera Up
Camera Front
vQuad2

Particle System.cpp (Render)

```
glUseProgram(program);

glUniform3f(glGetUniformLocation(program, "vQuad1"), vQuad1.x, vQuad1.y,
vQuad1.z);
glUniform3f(glGetUniformLocation(program, "vQuad2"), vQuad2.x, vQuad2.y,
vQuad2.z);

glUniformMatrix4fv(glGetUniformLocation(program, "vp"), 1, GL_FALSE,
glm::value_ptr(vp));

glActiveTexture(GL_TEXTURE0);
glUniform1i(glGetUniformLocation(program, "Texture"), 0);
glBindTexture(GL_TEXTURE_2D, texture);
```

Particle System.cpp (Render)

```
glEnable(GL_BLEND);
glDepthMask(GL_FALSE);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * vPosition.size(),
&vPosition[0], GL_STATIC_DRAW);

glBindVertexArray(vao);
glDrawArrays(GL_POINTS, 0, nParticles);

glBindVertexArray(0);

glDepthMask(GL_TRUE);
glDisable(GL_BLEND);
```

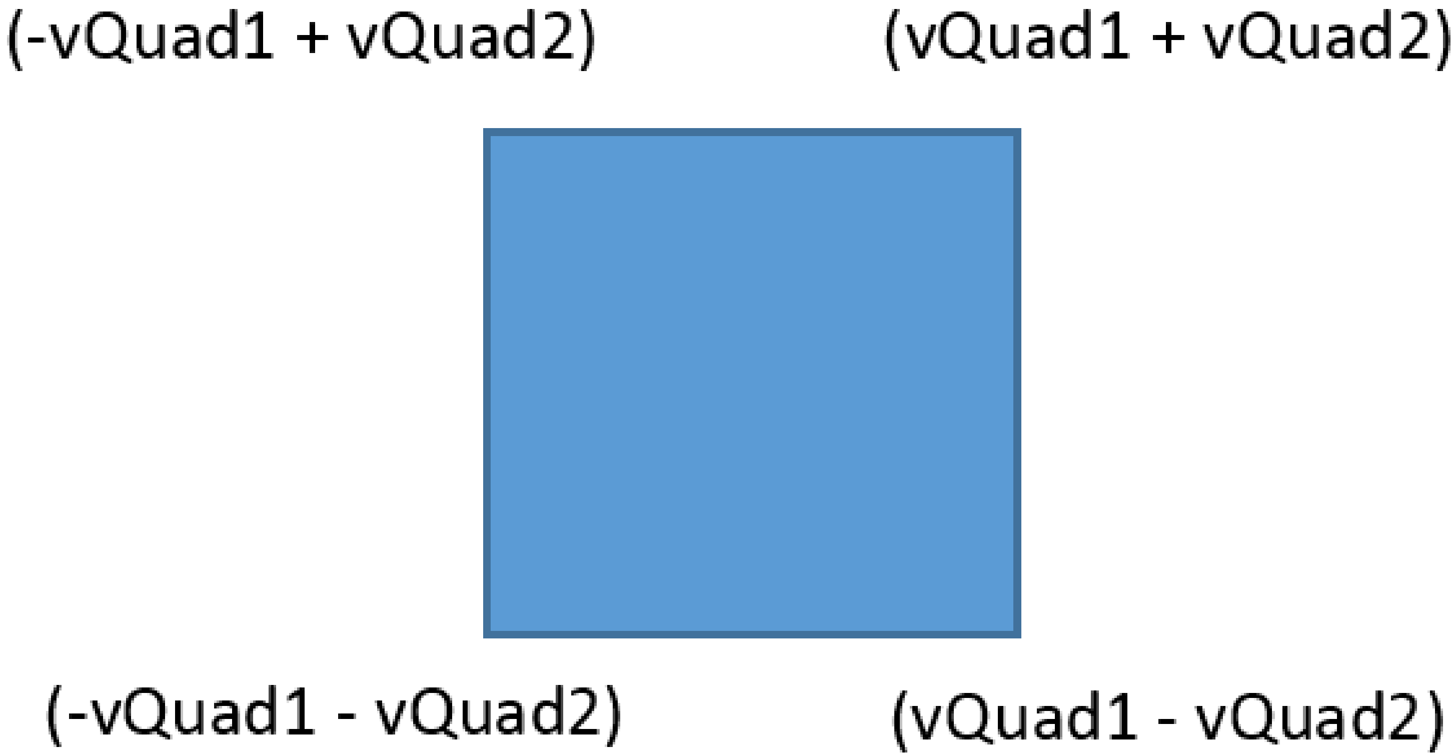
Vertex Shader

```
#version 330 core
layout (location = 0) in vec3 vertex;

void main(){
	// movement is in world space
	gl_Position = vec4(vertex, 1.0f);

}
```

Geometry shader

```
#version 330 core
layout (points) in;
layout (triangle_strip, max_vertices = 4) out;

uniform mat4 vp;
uniform vec3 vQuad1, vQuad2;

out GS_FS_VERTEX{
        vec2 texcoord;
}gs_out;

void main() {
    buildQuad(0.1, vp);
}
```

Geometry shader (contd.)



(-vQuad1 + vQuad2)

(vQuad1 + vQuad2)

(-vQuad1 - vQuad2)

(vQuad1 - vQuad2)

Geometry shader (contd.)

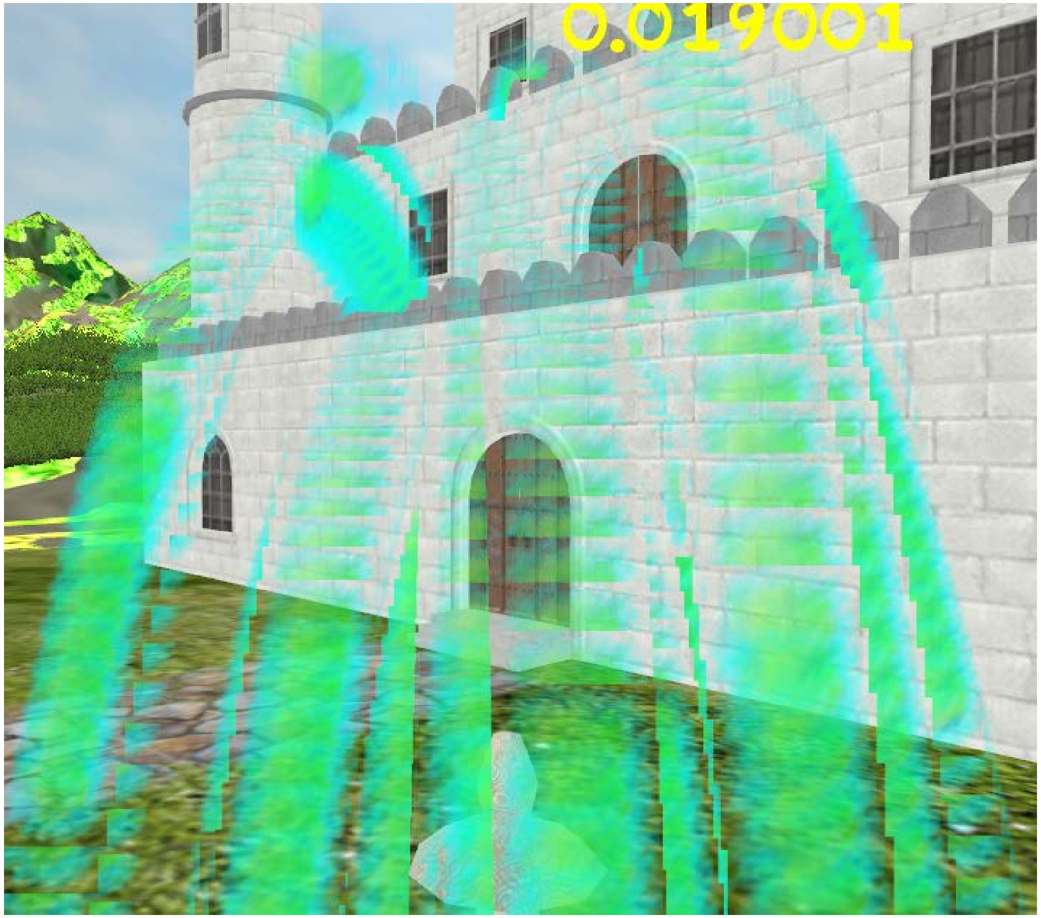
```
vec3 p1 =  gl_in[0].gl_Position.xyz +(-vQuad1-vQuad2)* size;
gl_Position = vp * vec4(p1, 1.0f);
gs_out.texcoord = vec2(0.0f, 0.0f); EmitVertex();

vec3 p2 =  gl_in[0].gl_Position.xyz + (-vQuad1+vQuad2)* size;
gl_Position = vp *  vec4(p2, 1.0f);
gs_out.texcoord = vec2(0.0f, 1.0f); EmitVertex();

vec3 p3 =  gl_in[0].gl_Position.xyz + (vQuad1-vQuad2)* size;
gl_Position = vp * vec4(p3, 1.0f);
gs_out.texcoord = vec2(1.0f, 0.0f); EmitVertex();

vec3 p4 =  gl_in[0].gl_Position.xyz + (vQuad1+vQuad2)* size;
gl_Position = vp * vec4(p4, 1.0f);
gs_out.texcoord = vec2(1.0f, 1.0f); EmitVertex();

EndPrimitive();
```

buildQuad function

Fragment Shader

```
#version 330 core

in GS_FS_VERTEX{
        vec2 texcoord;
}fs_in;

uniform sampler2D Texture;
out vec4 color;

void main(){

color =  texture(Texture, vec2(fs_in.texcoord.x , fs_in.texcoord.y)) *
        vec4(123.0f/255.0f, 173.0f/255.0f, 203.0f/255.0f, 1.0f);
}
```

Blending issue



Solution

- Sort each particle from back to front
- In the Particle class create a new float variable called cameraDistance.
- Pass the camera to the Particle class.
- In the update function store the distance to camera each frame for the particle.

```
this->position += velocity;
this->elapsedTime -= .000167;
this->cameraDist = glm::distance(this->camera->getCameraPosition(),this->position); //add
```

Solution

- In the particle system class, after updating the position of each particle.
- Sort the particle from far to near after updating the positon.

```
for (int i = 0; i < nParticles; i++) {
    particles[i].update(.0167);
    vPosition[i] = particles[i].getPosition();
}
std::sort(particles.begin(), particles.end(), myComparison);
```

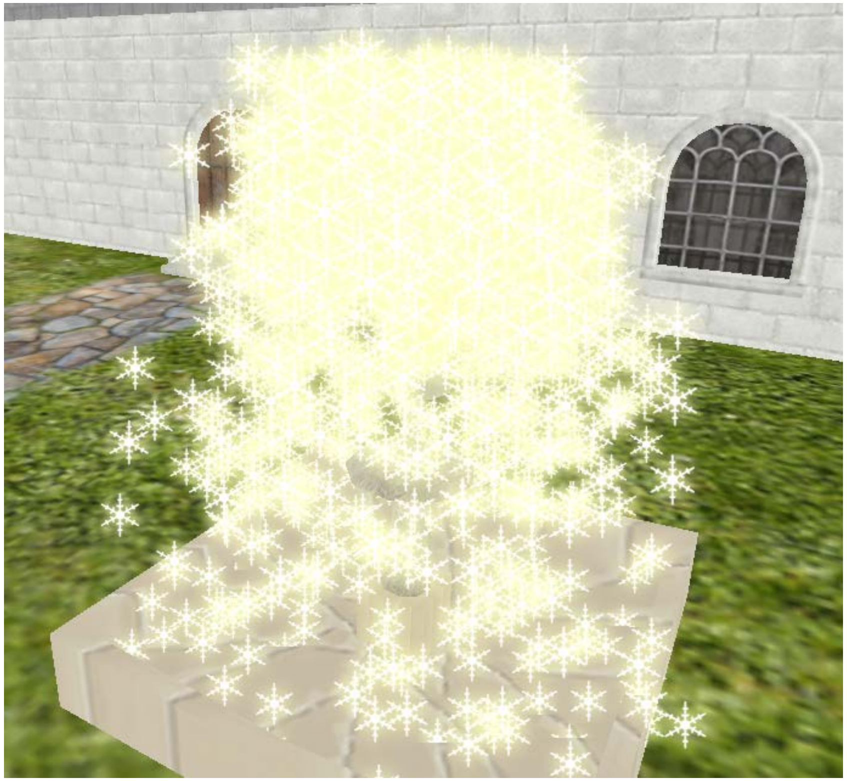
- Sorting comparison function

```
bool myComparison(Particle a, Particle b) {
    return (a.getDistToCamera() > b.getDistToCamera());
}
```

Solution

Solution

- Problem still slightly exists…. make the texture transparent.



Usage

- Init
  - particles = new ParticleSystem(glm::vec3(6.4f, 10.0f, 2.45f),camera, "Assets/images/particle.png");
- Render
  - particles->render(dt);

Notes

- The artifact-ing is still visible and it can be solved but it is beyond the scope of the class.
- You can create different particle types with different origin to create rain, smoke, fire, etc.
- This is a basic example you can add rotation and scaling to each particle
- You can also generate random colours for each particle
- Animate texture could be added.
- Replace the texture depending upon the stage o the lifecycle of the particle.

Excercise

- Using just GL_POINTS create a particle system (without geometry shader stage).
- Pass color attribute as well to specify color of each particle.
- Once particles are behaving as desired create billboarded particles using geometry shader.