CS7GV6 Computer Graphics Assignment 2

Van Allen Bruns Jr – 19329560

YouTube: https://youtu.be/nApxZNjOyEg

GitHub: https://github.com/brunsjrv/TCD-CG-Assignment2

Video Games and Stylized Rendering

For Assignment 2, my group selected video games, and more specifically stylized rendering, as our topic of exploration. For my portion of this exploration, I broke it down into five stages of implementation using the papers I selected (Mcguire & Fein, 2006) and (Reeves, 1983):

1. A particle movement simulation system
2. An object mesh representing each particle
3. Use of depth buffer
4. Outline generation
5. Cel-shading

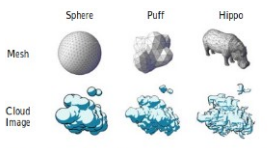
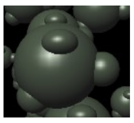


Figure 10. The billboard image used to render clouds is created from a mesh as a preprocessing step. *Left:* A sphere creates a bubbly cloud. *Center:* More complex meshes add small details. *Right:* Overly-detailed meshes create poor clouds.

I had proposed focusing on the first three stages, but ultimately implemented particle simulation, object mesh switching, and a basic cel-shader.

Object Mesh



This assignment was built using the existing framework developed for Assignment 1, modified towards this topic. Since my implementation of the first assignment included the ability to load mesh objects from Blender object files, it was straightforward adding different meshes. The mesh to the left is an example of a loaded mesh. Ultimately, I selected a basic IcoSphere from Blender for better performance and due to limited implementation time for selecting the best.

Cel-Shading

Modifying a cel-shader provided by one of my group members, Santiago, I added basic two-color cel-shading. This adds the cartoon-style rendering our group focused on.

```
#version 330

in vec3 vertexPosition;
in vec3 vertexNormal;

out vec3 fragPos;
out vec3 vsNormal;
out vec3 vsPosition;

uniform mat4 model, projection, view;
uniform vec3 lightPos;

void main(){
	fragPos = vec3(model * vec4(vertexPosition, 1.0));
	vsNormal = normalize(mat3(model) * vertexNormal);
	vsPosition = mat3(model) * vertexPosition;
	gl_Position =  projection * view * model * vec4 (vertexPosition, 1.0);
}
```

```
#version 330

in vec3 fragPos;
in vec3 vsNormal;
in vec3 vsPosition;

out vec4 fragColour;

uniform vec3 lightPosition;

void main() {
	// Global Lighting Variables
	vec3 lightDir = normalize(lightPosition - fragPos);
	vec3 norm = normalize(vsNormal);
	float intensity;
	vec4 color;

	intensity = dot(lightDir, normalize(norm));

	if (intensity > 0.5) {
		color = vec4(0.686, 0.933, 0.933, 1.0);
	}
	else {
		color = vec4(0.251, 0.878, 0.816, 1.0);
	}

	fragColour = color;
}
```

Particle Simulation System

The particle object framework developed is shown below. A particle manager handles the generation, removal, movement, and drawing of the particles in the system. Particles are added in conjunction with a particle portal (defining where particles enter the system) and a particle gun (defining how particles will move). Particles can be affected by multiple force fields, which can represent factors like wind affecting the movement of smoke particles.

```
Particle:
  void draw(mat4 view, mat4 projection)
  void affect(ForceField field)
  void advance()
  ParticleManager* pm
  GenericMesh* mesh
  vec3 position, direction, scale
  float speed
  int lifetime

ParticlePortal:
  vec3 createEntryPoint()
  int stepCount
  vec3 smallCorner,
      largeCorner

ParticleGun:
  vec3 createTrajectory()
  float createSpeed()
  vec3 initialDirection,
      rotation
  int stepCount
  float meanSpeed,
      varSpeedFactor
  int speedStep
```

```
ParticleManager:
  void add(ParticlePortal* portal, ParticleGun* gun)
  void clear()
  void remove(Particle& particle);
  void advance(list<ForceField>* forceFields)
  void draw(mat4 view, mat4 projection)
  GenericMesh* mesh
  list<Particle> particles, markedForRemoval
  vec3 scale
  int currentCycle, cycleWait, meanParts,
      varParts, meanLifetime, varLifetime

ForceField:
  bool isInField(vec3 position)
  vec3 affectDirection(vec3 direction,
                       vec3 position)
  vec3 direction, center,
      smallCorner, largeCorner
  float inward
```

The intention is a system generically written to allow for easy simulation of different particle environments with little implementation. The following code represents all required to simulate the environments detailed in the results section.

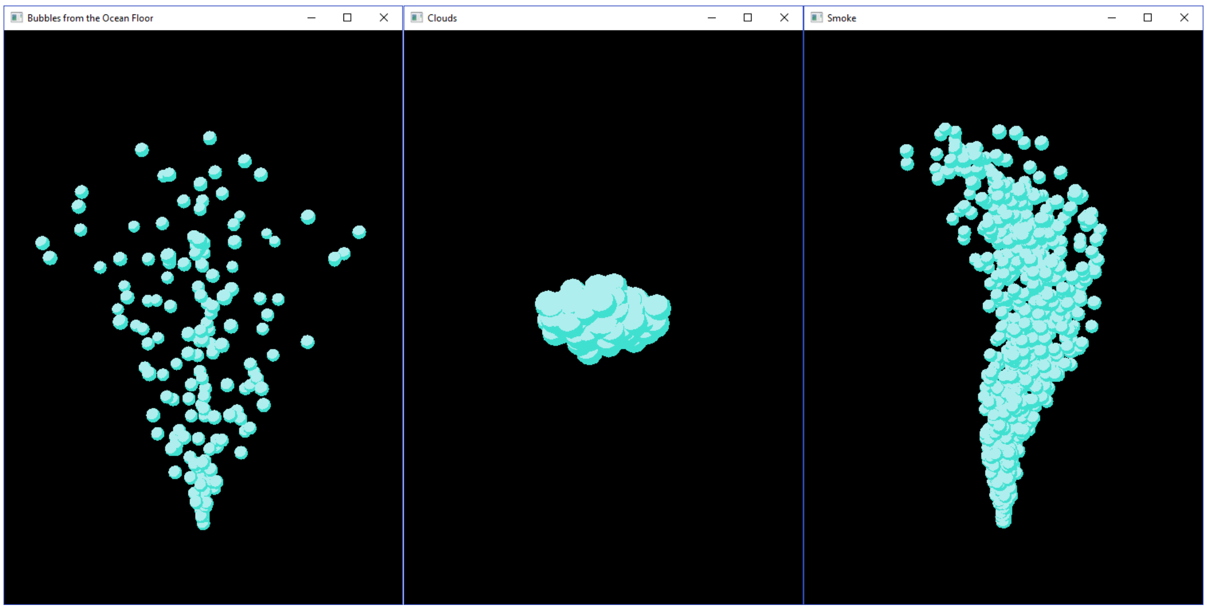
```cpp
ParticlePortal* portals[] = {
  // ParticlePortal(stepCount, smallCorner, largeCorner)
  // BubblePortal
  new ParticlePortal(5, vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 0.0f, 0.0f)),
  // CloudPortal
  new ParticlePortal(30, vec3(-4.0f, 14.0f, -2.0f), vec3(4.0f, 16.0f, 2.0f)),
  // SmokePortal
  new ParticlePortal(30, vec3(-0.05f, 0.0f, -0.05f), vec3(0.05f, 0.0f, 0.05f))
};
ParticleGun* guns[] = {
  // ParticleGun(initialDirection, stepCount, rotationRangeSize,
  //   meanSpeed, varSpeedFactor, speedStep)
  // BubbleGun
  new ParticleGun(vec3(0.0f,1.0f,0.0f),360,vec3(30.0f,360.0f,0.0f),0.015,0.0001,10),
  // CloudGun
  new ParticleGun(vec3(0.0f,0.0f,0.0f),360,vec3(360.0f,0.0f,360.0f),0.005,0.0001,10),
  // SmokeGun
  new ParticleGun(vec3(0.0f,1.0f,0.0f),360,vec3(15.0f,360.0f,0.0f),0.015,0.0001,10)
};
ParticleManager* managers[] = {
  // ParticleManager(shape, scale, cycleWait, meanNewParticles,
  //   varNewParticles, meanParticleLifetime, varParticleLifetime)
  // BubbleManager
  new ParticleManager(particleMesh, vec3(0.5f, 0.5f, 0.5f), 20, 2, 1, 1500, 500),
  // CloudManager
  new ParticleManager(particleMesh, vec3(1.0f, 1.0f, 1.0f), 10, 6, 3, 1500, 500),
  // SmokeManager
  new ParticleManager(particleMesh, vec3(0.5f, 0.5f, 0.5f), 20, 10, 3, 1500, 500)
};

// ForceField(direction, smallCorner, largeCorner, inwardFactor)

forceFieldLists[1]->push_back(ForceField(
              vec3(0.0f, 0.0f, 0.0f),
              vec3(-10.0f, 5.0f, -10.0f),
              vec3(10.0f, 25.0f, 10.0f), 0.0001f));
forceFieldLists[2]->push_back(ForceField(
              vec3(0.00002f, 0.0f, 0.0f),
              vec3(-10.0f, 5.0f, -10.0f),
              vec3(10.0f, 12.0f, 10.0f)));
forceFieldLists[2]->push_back(ForceField(
              vec3(-0.00002f, -0.00001f, 0.0f),
              vec3(10.0f, 25.0f, 10.0f)));
```

Results

Three simulations were produced using the particle system: bubbles, a cloud, and smoke (seen below). Parameters for each are also detailed below.



Bubbles

Single particle portal, producing particles at half scale. Particle gun pointing upwards in a 30° cone. Manager adding 2-3 new particles every 20 cycles, with particle lifetime of 1500-2000 cycles.

Cloud

Multiple particle portals in an 8x2x4 box, producing particles at full scale. Particle gun pointing outwards in a 360° sphere from each portal. Manager adding 6-9 new particles every 10 cycles, with particle lifetime of 1500-2000 cycles. Force field pulling particles back inwards towards the center.

Smoke

Multiple particle portals in a 0.1x0.1 square at the base, producing particles at half scale. Particle gun pointing upwards in a 15° cone. Manager adding 10-13 new particles every 20 cycles, with particle lifetime of 1500-2000 cycles. One force field pushing particles towards the right, one force field pushing particles towards the left and upwards after the first field.

Further Work

The original idea of using nailboards (2D objects with depth maps in a 3D world) wasn't completely realised. Performance improvements would be achieved if all particles were instead represented with 2D objects, especially as the number of particles increase. The simulation performance could also be further expanded through knowledge in real-time rendering.

Conclusions

This assignment gave me a more complete understanding of what it takes to implement a simulation framework. It also further defined what it means to change between different rendering styles. The group discussions and topic research helped in narrowing the scope, and practically staging the implementation.

References

Mcguire, M., & Fein, A. (2006). Real-time rendering of cartoon smoke and clouds. *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering - NPAR 06*.

Reeves, W. T. (1983). Particle systems---a technique for modeling a class of fuzzy objects. *ACM SIGGRAPH Computer Graphics*, 359-375.