COMP 3009 Project Report

"Campfire"

Course: COMP 3009
Date: 02/12/2012
Student Name: Luke Harper
Student Number: 100886836

Table of Contents

List of Figures

List of Tables

1. Introduction

When originally exploring projects, an interest that I came up with was environmental effects. I wanted to investigate and develop graphical components that took an arrangement of objects and turned it into a fully immersive scene. The three components that I identified with were lighting, texturing and animation. Particle effects in particular is One aspect of graphics that I felt both combined all these elements and is one of the best things that can be added to a scene to bring it to life. Particle effects like fire, smoke, fog, dust, and others really amplify an 'environmental feel.' So I then pursued what would be the most effective way of showing of a particle system while keeping true to the 'environmental feel' without being a strict tech demo. The scene I chose was a campfire. The particle effect would be front and center with nice distinct geometry. The Particle effect also represented the light source, and distributing objects around the fire could show off the lighting effects on the objects. Finally a skybox would wrap the scene providing continuity and fullness.

This project turned out quite well, the fire is lively and deliberate, and the rocks and logs around the fire seem to be in a futile attempt to contain the flames. The "billboards" of cats around the fire shows off the phong lighting and the skybox is a seamless night sky with surrounding mountains. The scene (minus the cats) looks like something that could be found in the forests of next to the Rocky Mountains.

2. Project

2.1 Objective of project

The objective of the project was to show off environment building techniques that can be used to take graphical objects and generate an atmosphere around a scene. The expected outcome was a believable campfire scene with a roaring fire.

My goal was to focus on all aspects that go into turning graphics into an environment, strategies and tactics used to infuse the scene with atmosphere and emotion.

2.2 Project Detailed Description

The main special feature of this project was to make the viewer believe they were sitting around a campfire.

This is accomplished with the campfire being front and center, with flames flickering around the fully textures logs and rocks. I emphasized the fire being the primary light source by making the colour of the light being the same as the fire particles. You see the reflection of the fire on the rocks around the campfire as they glow orange, and you can see the orange reflection on the ground as you face the fire. Additionally the billboards were constructed around the fire to further enforce the flame's effect on the scene. The Cat's colour on the billboard is slightly changed due to the colour of the flame, and the supports are very distinct in their colour change with the reflection of the fire. Finally the skybox envelops the scene in an endless a dark blue night sky illuminated only by the moon. The mountains wrap around the campfire 360 degrees naturally limiting your line of sight and effectively enforcing the focus on the campfire.

2.2.1 What special features were incorporated into the Project

I created a particle system in my scene to enhance the idea of "atmosphere" in order to create a proper environment for my scene. This was outside of the course content but I felt it was necessary to bring the scene alive. I added a particle system to my scene as I felt it was the best way to represent fire.

The fire particle effect adds the most character to my scene out of all the elements in my project, and is the main actor in my project. It is the only portion animated, and took the most amount of time. The fire effect allows my scene to come alive.

2.2.2 Special Elements

The special feature of my project was the particle system. The particle system is built by using the bill boarding technique in the geometry shader. The design for the particle system is distinct from a normal graphical object.

Architecture

The Particle System class is a derived class of the Graphical object, with overloaded createVAO and render functions.

- The createVAO function only uses the vertices, since the particle system is only points, we only need the location of the vertices, and no indices are required.
- The render function needs to render only points using the vertices, instead of the triangles using the indices that the standard Graphical Object uses. It also needs to handle particle specific rendering issues.

Geometry

The construction of the particles is done by specifying points in 3d space where each particle will be located; currently the different geometries supported to build the base layout of the particles are the sphere, and the cone geometry. The geometry builder samples random points along the geometry and specifies vertices at these locations. It then assigns the red channel in the colour vector as an id which with the particle will be animated and textured off of, and the system also uses the vertex's normal as the wander vector: the vector that the point will animate along in the vertex shader.

Billboarding

The particles in this project were built using the billboard technique. This has advantages and disadvantages.

The main advantage is that the particle building is mostly GPU based, limiting the impact on the CPU for processing the thousands of pixels. The fire system in this project is built out of 60,000 points, which if left on the CPU would turn into 60,000 quads that the CPU would have to pass into the GPU. The GPU also orientates the particle quads to face the viewer so the particles are always in view from every angle.

The main disadvantage is all of the animation and building of the particles is hidden from the rest of the project. The program on the CPU just sees the particle effect as static geometry, which is not an accurate representation to the animation and life cycle of the particles on the screen.

The billboard technique allows us to maximize the performance of our particle system, and to create nice looking effects with minimal effort.

Billboard particles are built using the following order:

- Define vertices in the program, which are passed to the shaders
- Have the vertex shader move the particles as desired, and pass that information to the geometry shader
- The geometry shader then takes those points and creates a quad centered around that point, it also supplies the texture co-ordinates for that specific quad, and passes the information to the fragment shader
- The fragment shader then adds the colour and texture to the appropriate fragment according to the information supplied by the geometry shader.

The Fire Effect

For this particular project, my goal was to create a fire particle system that would look the best as a campfire. I chose the cone geometry to be the base for my fire effect since it had two distinct advantages when considered for the campfire. The logs, when leaned together, formed a very pyramid/cone shape with each log leaning into each other forming a wide base and a small top. This is the shape that first came to mind when I thought of a campfire. The cone base for the fire effect happened to fit under the logs almost perfectly, filling the space under the logs. The second benefit to the cone geometry was that it seemed the most natural fire geometry out there. Fire likes to flicker and rise, but in a campfire it seems to rise and blend together, with the tallest peak of the fire being in the centre. The cone seemed to automatically give us this benefit.

Animation

In order to animate the particle effectively I used a particleID, and a wander vector. The particleID I passed through as the vertex's red channel in the colour vector, and the wander vector was passed through the normal vector. I used a sinusoidal function, using the particleID as the starting point on the sine function, the particles wander vertically up by the wander vector, and reset.

This allowed the particles to flutter up from the cone geometry making the effect that the flames were rising up form under the logs, and helped to remove the artificial structure of the cone geometry. The particles closer to the cone's point naturally rising higher, but each particle rising vertically allows the geometry to be hidden. The fire looks like it's swelling into the center.

Finally with a sinusoidal animation, particle lifespan and particle emitting is not an issue. When a particle's lifespan is over, it does not die, it simply resets. The particle lifespan is the time at which the particle needs to travel from its starting geometry to its apex, the random distribution of points, the variation in starting position, and varying distance of wander allows the particles to look like they are created, rise up, and die at the top.

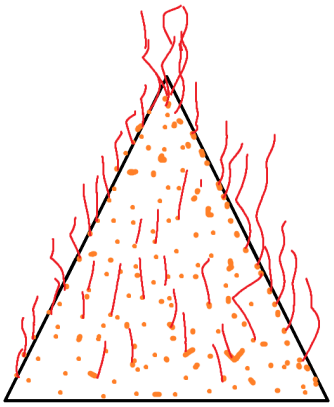

Figure 1: Animation of the Particles on a 2d representation of a cone

Rendering the Particles

In order for particles to be the most effective, they become different from other graphic object in two distinct ways, they need to be small, active, varying, and very numerous, as well as they need to be blended.

Texturing and Colouring

The fire effect is made up of 60,000 particles and using the particleID passed into the geometry shader takes picks a texture out of four variations, this allows variance in the fire texture, and with blending allows the particle to blend together, the colour is a constant colour made up of red and green to make a fire orange, which is combined with the colour of the texture in the fragment shader.

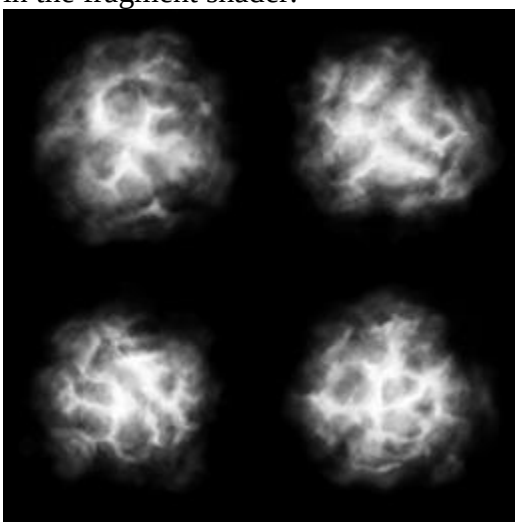

Figure 2: Particle variations used for individual fire particles

Blending

In order to correctly blend the particles, OpenGL needs to be in the correct state. First the project renders all of the graphical objects that in the scene. This allows the GL_DEPTH_TEST

to be correctly run when deciding if a particle should be drawn or not. After the objects are rendered, the particles can be rendered.

To render the particles, there are three steps. First we must disable the depth mask. This disables writing to the depth buffer; this allows openGL to do additive blending correctly regardless of depth order. Next turn on blending to be able to blend our particles together and finally to finish rendering the blend options must be specified. For Fire the blending that looked the best was glBlendFunc(GL_ONE_MINUS_SRC_ALPHA, GL_ONE) this blend caused the particles to vary in colour according to the intensity of the fire (number of particles overlapping). This turned the middle of the fire to yellow; similar to real fire in that the edges are always darker.

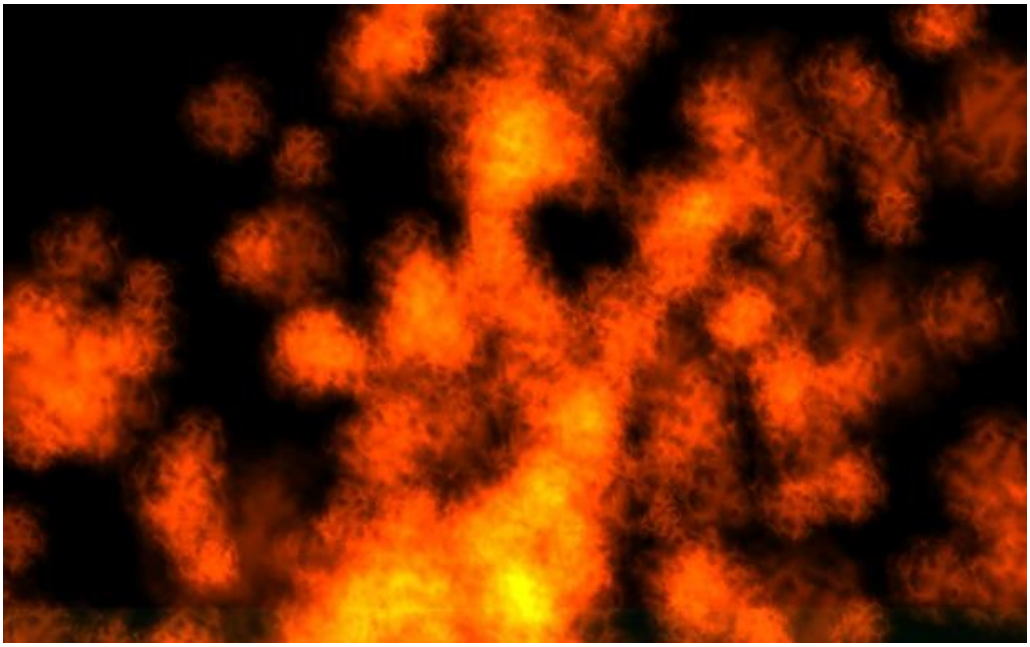

Figure 3: Blending of Fire Particles

The result of this Fire particle effect made the fire look alive and fierce, and really added to the atmosphere of the environment, and stands out in the scene.

2.3 External Resources

The Math behind the flame was not original; it comes from the Fire Demo by Oliver van Kaick, COMP 3501 2015

2.4 What was not accomplished

Three things were not accomplished within respect to the fire. The first enhancement was to make was pseudo-randomizing the size of the particles. This would be accomplished by using one of the remaining channels in the colour vector, either the green or the blue channels. The particles would either be a size of 0.5, 0.4, or 0.3 with a weighted chance for each size (planned weight was 65 % 0.3, 20% 0.4 and 15% 0.5. Despite the planning all being done, there was not enough time to accomplish this task. The second was variation of the light intensity according to the same function that the sinusoidal animation was using. However the animation and the intensity never quite matched up so it was not implemented. Finally, a stretch goal was to adjust

the light vector according to the animation of the particles; however this was a planned extension of the variation in light intensity and since the variation of light was never finalized this was never pursued.

2.5 What was hard

The hardest part of this project was the particle system. Since a particle system in OpenGL was never pursued in the scope of the course, building the desired particle system was a lot slower than planned. Conceptually a particle system is simple enough, and planning for the particle system was relatively straight forward. Minor bugs however took exponentially longer to resolve, since it was unknown if something not working meant that the implementation was wrong, or if it was a minor bug like a typo or wrong variable name. The previous knowledge base of how OpenGL works was not there in order to quickly diagnose a problem, so manual testing and tweaking had to be followed in order to identify and diagnose a problem.

Finally the blending was particularly tricky as since that had been the last thing taught in the course, it was also the aspect of OpenGL that was explored the least. Trial and error was accomplished for the blending however a couple days were lost trying to figure out how the depth mask work. The confusion was solved when the ordering of the object being drawn was the solution (drawing normal objects first, and particles last)

3. Project Work

This section shall describe how you accomplished the project:

- The Agile method of development was used, Small iterative tasks accomplished to larger tasks. Ex: build a cylinder, texture + normal on cylinder, build a large object with the small objects. Etc.
    - Note: Being a 1 man process, the developmental cycle was way more fluid as no need to integrate with other people code / update people on progress

Task	time Estimates	Actual Time	Reasons
Overall effort	60 Hours	75 Hours	Problems with particle effect implementation and blending resulted in longer implementation and integration
Researching and Design of Project	5 Hours	5 Hours	Pretty clear idea of what project I wanted, just could not decide on the objects around my scene
Design of code	25 Hours	10 Hours	Ended up adapting the demo/project code to the project so half the design was already done
Implementation	20 Hours	40 Hours	Particle effects took way longer to research and understand
Integration and testing	10 Hours	20 Hours	Figuring out the Blend took while

Table 1: Project Schedule - this table presents the estimated work time vs. the actual time that it took to complete the tasks.

4. Grade

I feel like I deserve an 85% on this project. Accordingly to course material, I implemented most of the major features in the course (hierarchical objects / transformations, 3-term lighting, texture mapping, skyboxes), the software architecture is decently planned on nicely with room to expand (the particle effect shader is very specific to fire animation but can be expanded on), Additionally the special feature of a fire particle effect was outside the scope of this course and built on the ideas of blending, the VAO, vertex shader animation, and texturing, while exploring completely new concepts of geometry shader (albeit very simplified). I feel like I do not deserve higher because the particle effect itself is relatively simple (cone geometry expanding upwardly), although this was done both as a time constraint and to effectively show others the ease at which implementing particle systems of their own is possible. Additionally, I was unable to effectively dynamically update the lighting and surrounding geometry in accordance with the particle effect.

5. Conclusions

I think the project turned out to be quite the success, the fire looks pretty realistic and alive, and the skybox wraps the scene nicely. I learned a couple of lessons about OpenGL and coding new concepts in general. First, OpenGL is just a giant state machine and to not be afraid to manipulate the state machine where appropriate. Initially I only wanted to manipulate OpenGL in the solution class / solution's render or update functions. However it is way more effective to manipulate OpenGL on demand in different type of object's render functions. One final thing in OpenGL is always undo any states that were set at the end of a function, this sets OpenGL back to its initial state and removes any weird layover affects that may exist if you forget to reset. I also learned when researching a new topic, is make sure you have a good understanding of what you have just implemented before moving on, initially I would figure out how to implement the desired piece and then move on trying to maximize time. This was detrimental because later on in the life cycle if I implemented something and it did not work, or accidently affected the previous piece, I would not fully understand how those two elements interact with each other because I did not understand the original piece enough. If I could do the project over again I would do more research planning. In this current project I researched the specific step I was on, for example; I would research billboard shaders and then implement them. If I could redo this again, I would research the billboard particle effects and then research how to do blending before I started implementing both, this is because I wasted some time going back and changing old code because(using the blending example again) the ordering mattered in how things were rendered. If I had time to continue working, I would continue playing with dynamically updating the lighting in accordance to the flame, grow the flame with different geometry particle effects to further increase the natural look of the fire, and make more environment appropriate models surrounding the fire.