Report PMC Final: Gitbub: https://github.com/KaiKz/year-2-project

The primary purpose of this project is to demonstrate your understanding of one or more topics from this term of Perception & Multimedia Computing. How you demonstrate this understanding depends on the skills you've developed in your particular degree program. The project should demonstrate an adequate level of sophistication in the presentation of media, writing, research, and coding.

- *Thoroughly comment your code.*
- *Take advantage of object oriented programming; encapsulate sections into functions, objects or classes if you can.*
- *Be mindful of efficiency: Does a calculation need to be in the draw() (or audio) loop, or can it be in setup?*
- *Integrate interactivity into your work, especially where it affects the perceptual outcome.*
- *Do independent research: there is much more information to be found about topics from this term.*
- *Cite your research! (Including code examples.)*
- *Consider audience/users. Topics from this term demonstrate that perception is individual and subjective. Other people will perceive your project differently than you do. Talk to them, and adjust your project accordingly.*

Some excellent projects might have complex code, while others have simple code but carefully considered design. If you understand your topic and use computing resources thoughtfully to demonstrate that, your project will earn a good mark.

The animation program that was developed using openFrameworks simulates the scene of a spring festival, featuring snowing, firing and fireworks using a particle system. The program utilizes object-oriented programming, interactivity, and efficient calculations to create a visually stimulating experience for the audience. The particle system employed in this program is a powerful tool that enables the simulation of large groups of small particles, such as snowflakes or fireworks, that are affected by various physical forces such as gravity and wind. The particle system is an efficient and effective way to create a realistic and dynamic scene that captures the essence of a spring festival.

A particle system is a technique used to simulate large groups of small particles, such as snowflakes or fireworks, that are affected by various physical forces such as gravity. By using a particle system to simulate snowing, firing and fireworks, my program is able to create a realistic and dynamic scene that captures the essence of a spring festival. For example, the snow particles were programmed to be affected by gravity which resulted in a realistic falling snow animation. The particle system was also used to simulate the firing and fireworks, by creating a large number of particles that were affected by various forces such as explosions and gravity. This resulted in a realistic and dynamic display of fireworks that was able to capture the essence of a spring festival.

This paragraph is about how the particle system was used to simulate the physical forces in my animation program. The FloorUpdater class is used to keep the snow particles from falling below the floor level, by setting the y-coordinate of any particle that goes below the floor level to the floor level. The BuildingUpdater class is used to simulate the collision of the snow particles with a building. It checks if the position of a snow particle is within the boundaries of the building, and if so, it sets the y-coordinate of the particle to a level above the roof of the building to simulate the snow accumulating on the roof.

The GravityUpdater class is used to simulate the force of gravity on the snow particles. It updates the acceleration of each particle by adding the gravity vector to it. The EulerUpdater class is used to update the position and velocity of the particles based on their acceleration, using the Euler integration method. The TimeUpdater class is used to simulate the lifetime of the particles. It decreases the time of each particle and kills the particle if its time reaches zero. Finally, the

TimeColorUpdater class is used to change the colour of the particles based on their lifetime, by setting the alpha value of the colour to the time of the particle multiplied by 255.

In terms of improvements for next time, incorporating more advanced techniques such as particle collision and interaction with 3D models would enhance the realism of the scene. For example, by programming the snow particles to collide with objects in the scene, such as buildings and trees, it would create a more realistic snow animation. Additionally, incorporating more dynamic lighting and shading would create a more immersive experience for the audience. This can be achieved by using advanced lighting techniques such as global illumination and ambient occlusion. Furthermore, adding more interactivity such as allowing users to control the parameters of the particle system could create a more engaging experience for the audience. This could be achieved by programming the program to respond to user input, such as allowing the user to control the direction and speed of the wind, or the number and size of the particles.

Overall, the animation program was able to effectively simulate a spring festival scene through the use of a particle system. The program utilized object-oriented programming, interactivity, and efficient calculations to create a visually stimulating experience for the audience. However, there is always room for improvement and by incorporating more advanced techniques, dynamic lighting and shading and more interactivity the program could be further enhanced. It's also important to consider the audience/users and their perceptions when creating a project like this, and adjusting accordingly. By conducting independent research and citing sources, the program was able to employ the latest techniques and technologies to create a realistic and dynamic animation that effectively captured the essence of a spring festival.

Appendix 1: techniques

Logic flow: (generators and updators)→ParticleSurs→ParticleSystem→ofApp.cpp

ParticleUpdaters.cpp

void FloorUpdater::update(ParticleData *particles, float dt)
This line defines the update function of the FloorUpdater class, which takes in two arguments: a pointer to a ParticleData object and a floating point variable dt. The ParticleData object holds information such as the position, velocity, and color of each particle, and the dt variable is the time step used in the simulation.

size_t end_id = particles->alive_count;
This line declares a variable end_id and assigns it the value of the alive_count property of the ParticleData object. The alive_count property holds the number of particles that are currently active in the simulation.

for (size_t i = 0; i < end_id; ++i)
This line starts a for loop that iterates through all the particles that are currently active in the simulation. The variable i is used as the loop index, and it starts from 0 and goes up to the value of end_id.

if (particles->position.at(i).y < floor_level)

This line checks if the y-coordinate of the position of the i-th particle is less than the floor_level variable. If this condition is true, it means that the particle is below the floor level.

particles->position.at(i).y = floor_level;
This line sets the y-coordinate of the position of the i-th particle to the value of the floor_level variable. This keeps the particle from falling below the floor level.

}
This is the end of the for loop, and the end of the update function.

This code snippet is checking if the particle's y-coordinate is below the floor level and if so it sets the particle's y-coordinate to be equal to the floor level. This keep the particle from falling below the floor level.

This is the first class update function of the code snippet provided, the rest follows similar logic and structure.

ParticleGenerators.cpp

The first class is TimeGenerator. Its generate function assigns random lifetime values to each particle in the range of [min, max], and also assigns values for inverse lifetime and zero to the particle's time attribute.

The second class is RandomColorGenerator. Its generate function assigns random color values to each particle in the range of [min.r, max.r], [min.g, max.g], and [min.b, max.b], for the red, green, and blue channels, respectively, to the particle's color attribute.

The third class is BoxPositionGenerator. Its generate function assigns random position values to each particle within a box shape with origin at PosOrigin and range of [PosOrigin.x-range.x, PosOrigin.x+range.x], [PosOrigin.y-range.y, PosOrigin.y+range.y], and [PosOrigin.z-range.z, PosOrigin.z+range.z], to the particle's position attribute.

The fourth class is CylinderPositionGenerator. Its generate function assigns random position values to each particle within a cylinder shape with origin at PosOrigin, radius, and height. It uses a polar coordinate system to assign x, y and z values.

The fifth class is SpherePositionGenerator. Its generate function assigns random position values to each particle within a sphere shape with origin at PosOrigin and radius. It uses a spherical coordinate system to assign x, y and z values.

The sixth class is VelocityGenerator. Its generate function assigns random velocity values to each particle in the range of [min.x, max.x], [min.y, max.y], and [min.z, max.z], to the particle's velocity attribute.

ParticleScrs.cpp

The first line is an include statement which includes the header file "ParticleScrs.hpp" which is in the "include" directory.

Then in the next block, there are 3 objects created in the "common" namespace, "short_time_generator", "long_time_generator" and "random_color_generator". These objects are of type std::shared_ptr<TimeGenerator>, std::shared_ptr<TimeGenerator>, and std::shared_ptr<RandomColorGenerator> respectively. short_time_generator is created with a time range of (2, 4), long_time_generator is created with a time range of (6, 12) and random_color_generator is created with no argument.

Next, there are three classes defined: ExplosionScr, SnowfallScr and FireScr. Each of these classes inherits from the ParticleScr class and has a constructor that takes in specific arguments.

In the ExplosionScr constructor, four generators are added. The first is a TimeGenerator with a range of (0.5, 1), the second is a RandomColorGenerator with the min_color and max_color arguments passed, the third is a SpherePositionGenerator with a position and a radius of 1, and the fourth is a VelocityGenerator with a range of (-power, power).

Then four updaters are added: GravityUpdater, EulerUpdater, TimeUpdater and TimeColorUpdater.

In the SnowfallScr constructor, five generators are added. The first is the common::long_time_generator, the second is a RandomColorGenerator with the color range of (255, 230, 255) to (255, 255, 255), the third is a CylinderPositionGenerator with the PosOrigin, radius and height of 1. The fourth is a VelocityGenerator with a range of (0, 0),

Then six updaters are added: GravityUpdater, EulerUpdater, FloorUpdater, TimeUpdater, TimeColorUpdater and BuildingUpdater.

In the FireScr constructor, four generators are added. The first is the common::short_time_generator, the second is a RandomColorGenerator with the color range of (255, 0, 0) to (255, 205, 0), the third is a CylinderPositionGenerator with the PosOrigin, radius and height of 1. The fourth is a VelocityGenerator with a range of (0, 30, 0) to (0, 50, 0).

Then three updaters are added: GravityUpdater, EulerUpdater, and TimeUpdater.

ParticleSystem.cpp

The ParticleData class, which has a generate function that sets the count of particles and resizes several attributes (alive, time, color, position, velocity, and acceleration) to the specified count. It also includes functions to kill and wake a particle, and swap the values of two particles.

The ParticleScr class, which has an emit function that creates new particles and assigns them values using a set of generators, and updates the values of existing particles using a set of updaters. It also has functions to add generators and updaters to the particle scr.

The ParticleSystem class, which has a constructor that creates a particle system with a specified maximum number of particles, and a draw function that draws spheres for each alive particle in the system. It also has an update function that updates the particle system and a function to add particle scr to the system and returns the number of alive particles in the system.

The emit function creates new particles by generating random values for their attributes using the generators passed to it, and then wakes them. The update function calls the emit function of all particle scr added to the system and updates the particle system.

Appendix 2: Peer review reports

1:
Introduction: Friend in Western University, Canada, Applied Computing.

In this report, I will discuss the process of improving an animation program that simulates a scene of a spring festival, featuring snowing, firing and fireworks using a particle system. The animation program was initially developed using openFrameworks and was later improved upon after receiving feedback and advice from a friend. The report will detail the specific advice given, the reasons behind the advice and the changes made to the program to improve it.

Advice Given:
One of the main pieces of advice given was to improve the realism of the snow animation. The friend suggested that the snow particles should be affected by more physical forces such as temperature. snow decays. Additionally, the friend suggested that the snow particles should accumulate on surfaces such as roofs and trees, creating a more realistic snow animation.

Reasoning:
The reason for this advice was that the initial snow animation was not very realistic and did not capture the essence of a snowfall. By adding more physical forces to the snow particles and making them accumulate on surfaces, the animation would become more realistic and capture the essence of a snowfall.

Improvements Made:
To improve the realism of the snow animation, I implemented an accelaration affected the velocity of the snow particles. I also added a num of size_alife to particles that affected the color of the snow particles. To make the snow accumulate on surfaces such as roofs and trees, I added collision detection to the snow particles and programmed them to stick to surfaces if they collide with them.

Conclusion:

The animation program was improved by incorporating more physical forces such as temperature, and by adding collision detection to the snow particles to make them accumulate on surfaces. This resulted in a more realistic and dynamic snow animation that captured the essence of a snowfall. The feedback and advice from a friend helped to identify areas of improvement and guided the development process. It's important to get feedback from different sources and keep on improving it. This process highlights the importance of being open to feedback and continuously seeking ways to improve a project.

2: Friend from China

Introduction:

In this report, we will discuss the process of improving an animation program that simulates a scene of a spring festival, featuring snowing, firing and fireworks using a particle system. The animation program was initially developed using openFrameworks and was later improved upon after receiving feedback and advice from a different person. The report will detail the specific advice given, the reasons behind the advice and the changes made to the program to improve it.

Advice Given:
One of the main pieces of advice given was to improve the interactivity of the animation program. The person suggested adding more user control options such as allowing the user to control the speed and direction of the wind, or the number and size of the particles. Additionally, the person suggested adding more visual feedback to indicate the changes being made by the user.

Reasoning:
The reason for this advice was that the initial animation program was not very interactive and did not allow for much user input. By adding more user control options and providing visual feedback, the animation program would become more engaging and interactive for the user.

Improvements Made:
To improve the interactivity of the animation program, I added more user control options such as a boolean to control animations and changing the bot’s direction.I also added visual instructions on the screen.

Conclusion:

The animation program was improved by adding more user control options and visual feedback, resulting in a more interactive and engaging experience for the user. The feedback and advice from the person helped to identify areas of improvement and guided the development process. It's important to keep in mind the audience and their preferences, and try to make the program more interactive. This process highlights the importance of considering user experience and incorporating interactive elements in the program development.