Report

Assignment-3

Team members' details

- Team member 1: Abhirami R Iyer
  Roll No: 112201001

- Team member 2: Nandhana Sunil
  Roll No: 112201008

Codebase

1. Creating spheres
   Spherical models are created by giving inputs as the number of segments and sectors the sphere should have. The `generateSphere` function generates the vertices, base colors for spheres and its normals into a vector. Then VAOs and VBOs are setup for these spheres and corresponding data are sent to these buffers.
2. Giving textures
   The `loadtexture` loads the texture given as an argument to this method. The `draw` method renders the sphere with the appropriate texture.

3. Rotation of sphere
   Rotation angle of each of the spheres are updated in each render loop. Using the `rotate` method from glm, the sphere models are rotated. This is applied to rotate the models of Sun, Earth and Moon.
4. Revolution of Earth and Moon.
   Revolution of Earth and Moon and other planets is attained through small incremental translations of earth and Moon and other planets in their respective elliptical orbits.

------ similar to assignment - 2 --------

When it comes to supernova explosion :

1. The explosion rendering code : corona_effect.cpp, corona_effect.hpp and particle_system.hpp, particle_system.cpp
2. The explosion constants like time and radius etc : config.hpp

3. The main rendering code : main.cpp (with functions to initialise planet models, their buffers, initialise the explosion components and update the supernova simulation, calculate the bezier curve and render them)

IMPLEMENTATION DETAILS

The corona layer, was rendered using the corona class : defined in corona_effect.hpp and corona_effect.cpp

The corona effect is rendered using textured quads with many layers rotated and scaled so that we could get a feel of glowing.

These quads are rendered using billboarding technique (which ensures a flat object always faces the camera).

The explosion is done using particle_system.hpp and particle_system.cpp, the particles are rendered as GL_POINTS where each point is expanded to a textured quad by the fragment shader. The particles decay after a time. A turbulence is also added to the particle using the stb_perlin.h. The particle color evolves over time (measured with life) and gets faded over its life.

The supernova is divided into states :

- STATE_PRE_EXPLOSION
- STATE_CORONA_EXPANDING
- STATE_EXPLODING
- STATE_FADING

In the STATE_PRE_EXPLOSION phase, the sun (its normal radius) exists with it's corona.

In the STATE_CORONA_EXPANDING (which it enters on pressing S) → The sun grows to match the initial effect of corona explosion i.e, The Sun sphere starts expanding from sun_radius towards CORONA_BASE_SIZE. The Corona effect starts expanding from CORONA_BASE_SIZE towards maxSize. Calculating the time between the pressing of S and the particles i.e., CORONA_TO_PARTICLE_DELAY it smoothly expands(using cubic expansion curve) the sun radius to an increased desirable radius which is 1 unit less than the corona radius, as corona would be visible only then.

In the STATE_EXPLODING state (the sun is not rendered) and mercury and venus within a certain delay would stop getting rendered.

In the STATE_FADING state : the corona particles start dimming from the initial brightness to 0.

i.e., the billboard quads, begin to diminish in size and brightness.

The fading time is calculated by :

$$t_{fade} = (currentTime - (coronaStartTime + coronaParticleDelay + explosionTransitionDuration)) \div coronaActualFadeDuration$$

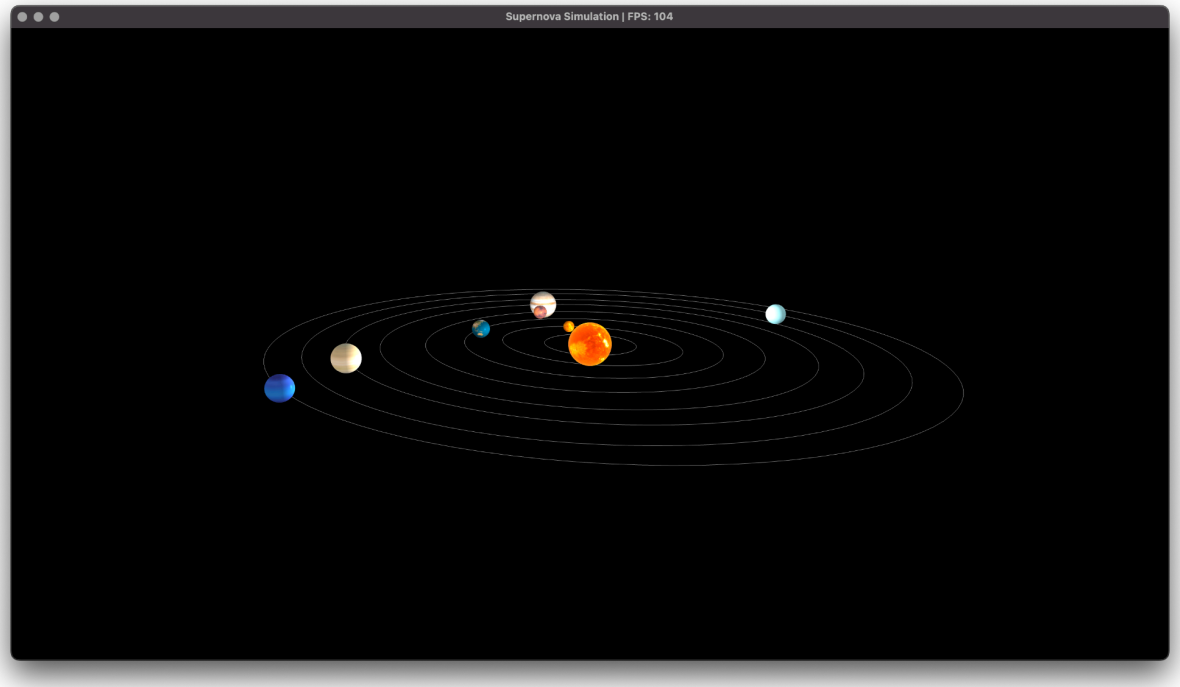
Clamped to [0,1] and then the power curve $t^{coronafadespeed}_{fade}$, interpolated with both size and brightness. Particles life also dissipates based on the equation $\Delta life = \Delta t / lifetime$. Once the life is less than 0, its alpha is forced to 0 (thereby rendering it fully transparent).

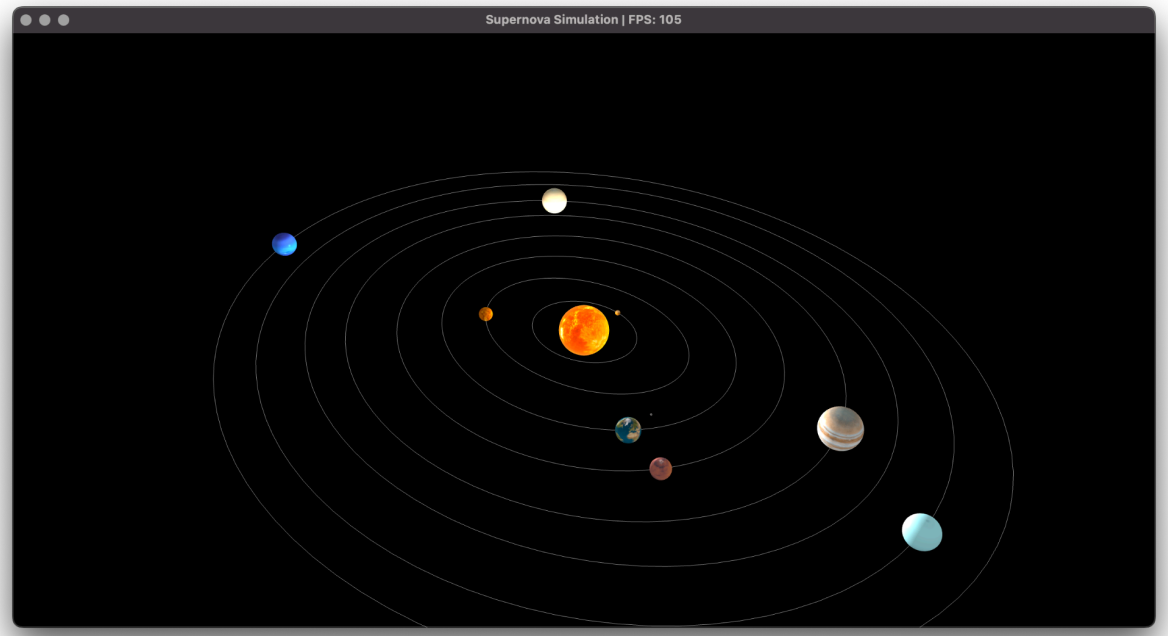
Shortly after the explosion we'll stop rendering Mercury and Venus based on their respective delays.

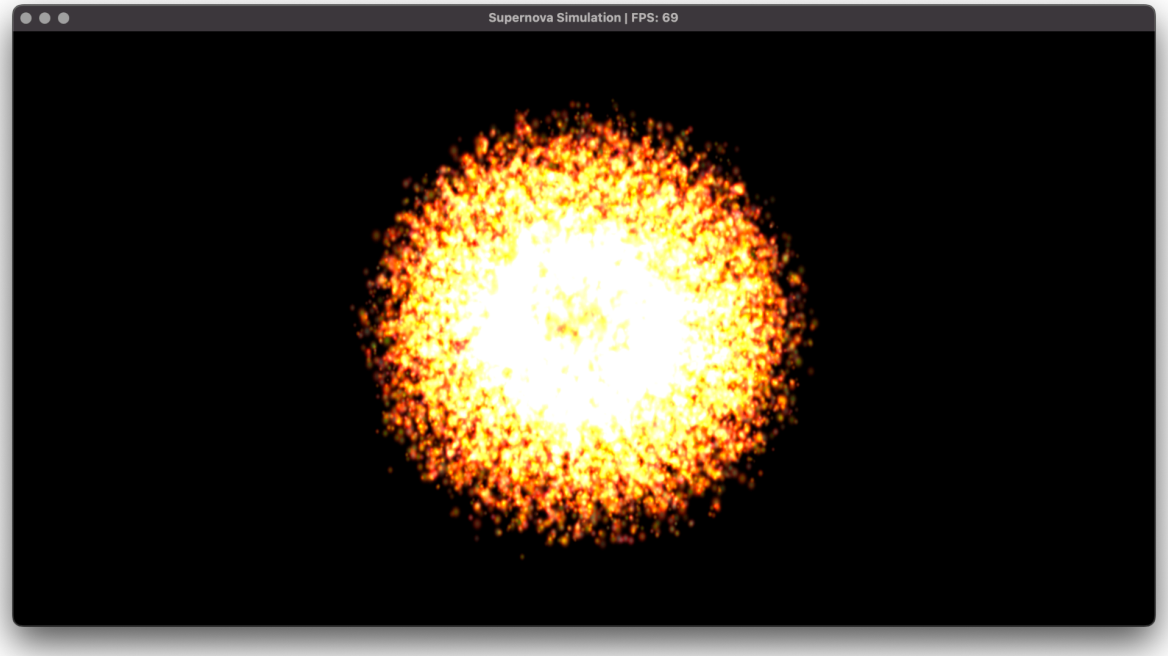
The non-gravitational movements of the planets are done using a cubic bezier curve calculation.
The curve widths are varied for planets keeping the equation same.

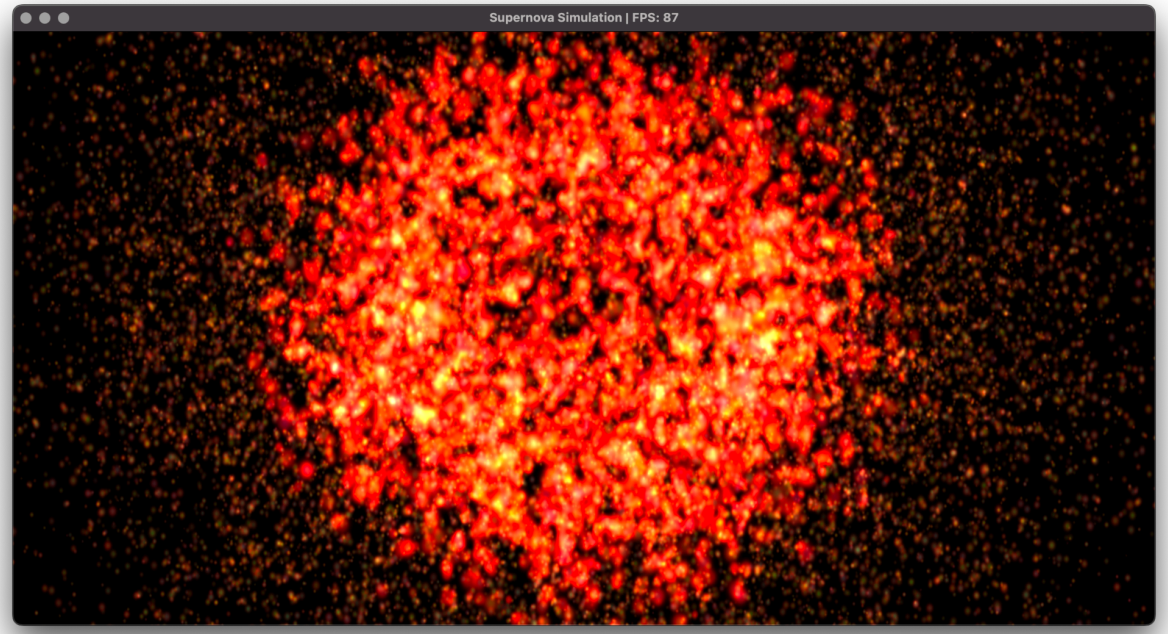
$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t \cdot P_1 + 3(1-t)t^2 \cdot P_2 + t^3 P_3$$

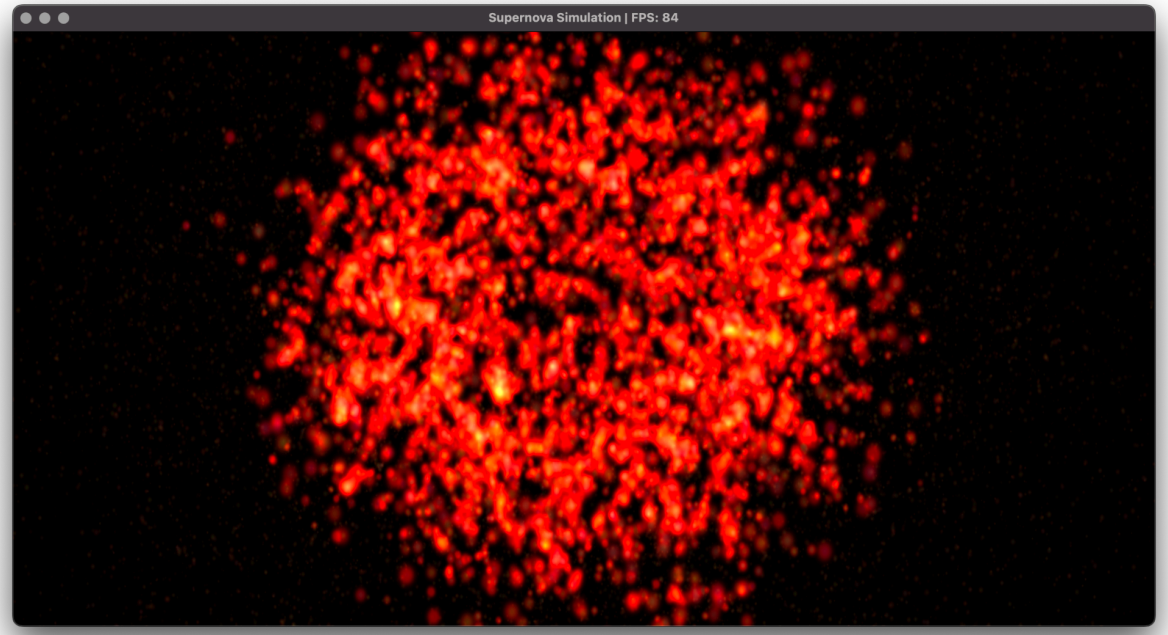
The camera animation is done by initialising camera paths, setting camera positions and timings for their translation. Initially the path is initialised in such a way that it looks like a tour around the solar system. When the S key is pressed to trigger the supernova explosion, the camera path is set to zoom into the explosion remains.

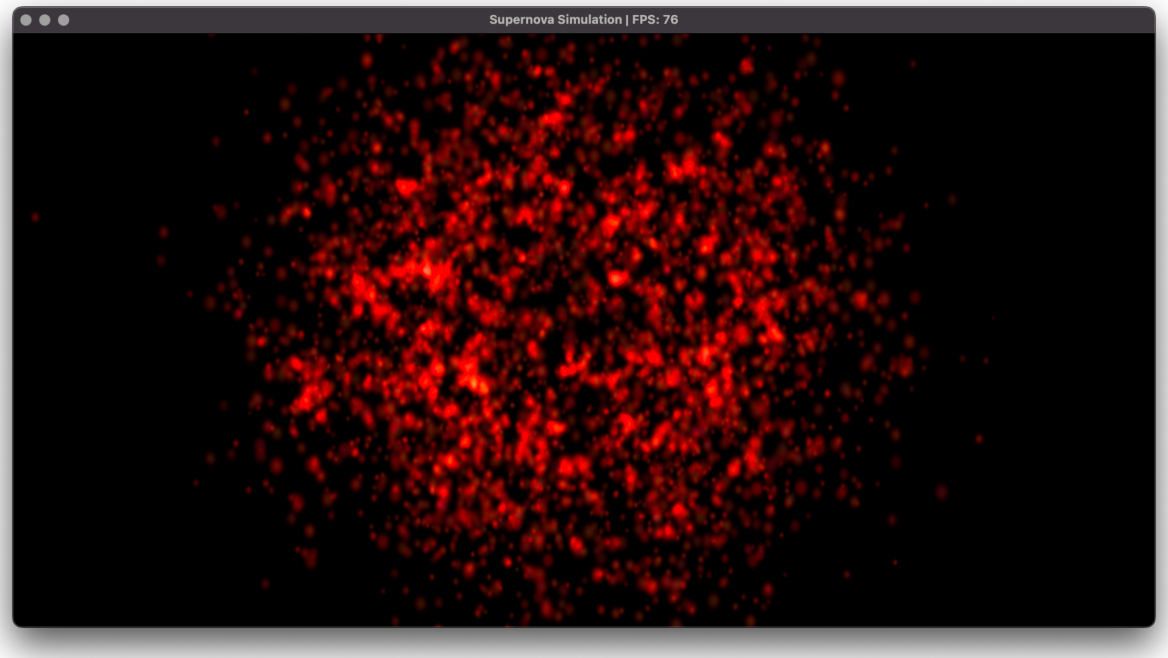


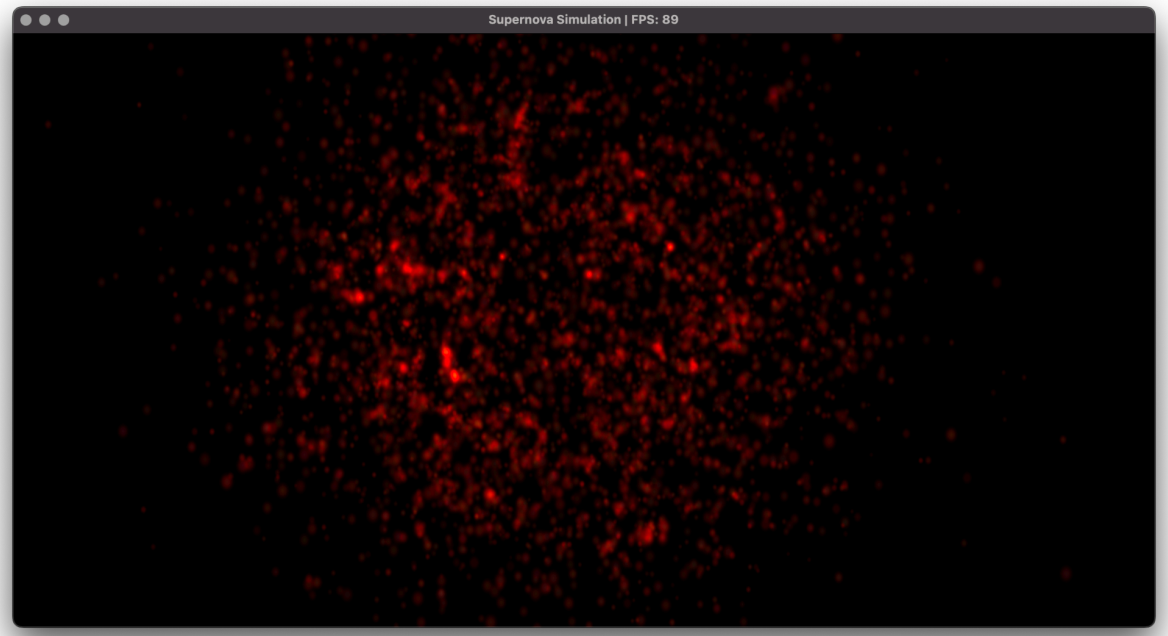

Supernova Simulation | FPS: 105

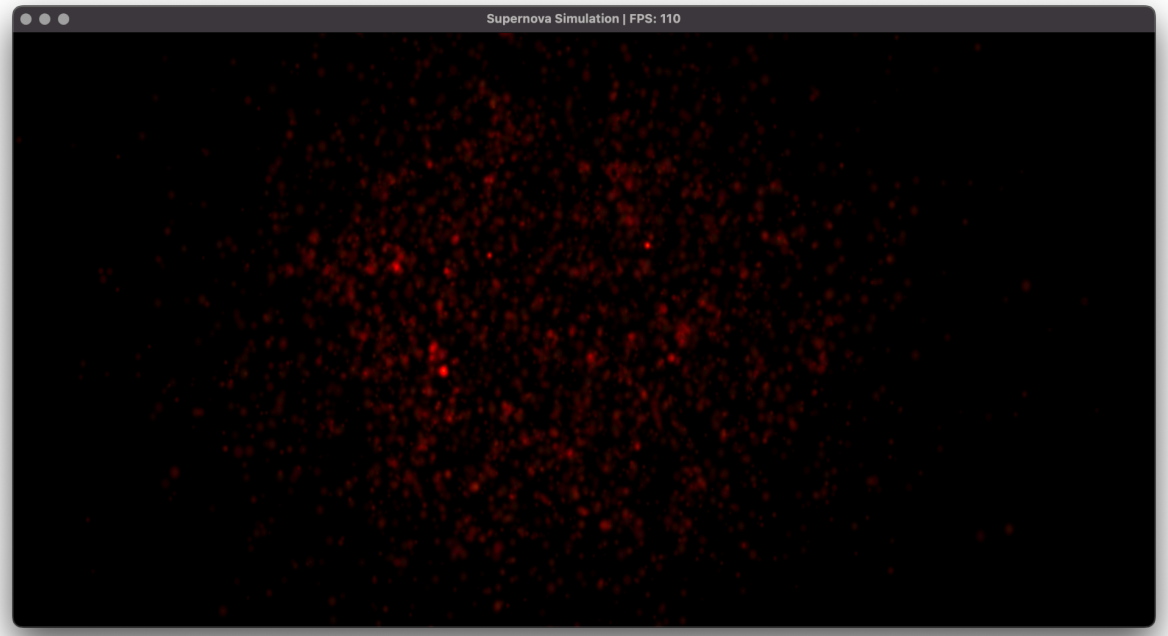

Supernova Simulation | FPS: 69


Supernova Simulation | FPS: 87


Supernova Simulation | FPS: 84


Supernova Simulation | FPS: 76


Supernova Simulation | FPS: 89


Supernova Simulation | FPS: 110