User Interface for Particle Controller

by

Nurunnisa Abdul Aziz

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Technology (Hons)
(Information Technology)

JULY 2005

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

User Interface for Particle Controller

by

Nurunnisa Abdul Aziz

A project dissertation submitted to the
Information Technology Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
BACHELOR OF TECHNOLOGY (Hons)
(INFORMATION TECHNOLOGY)

Approved by, [illegible]

(Mr Mohamed Nordin Zakaria)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
July 2005

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources of persons.

[illegible]

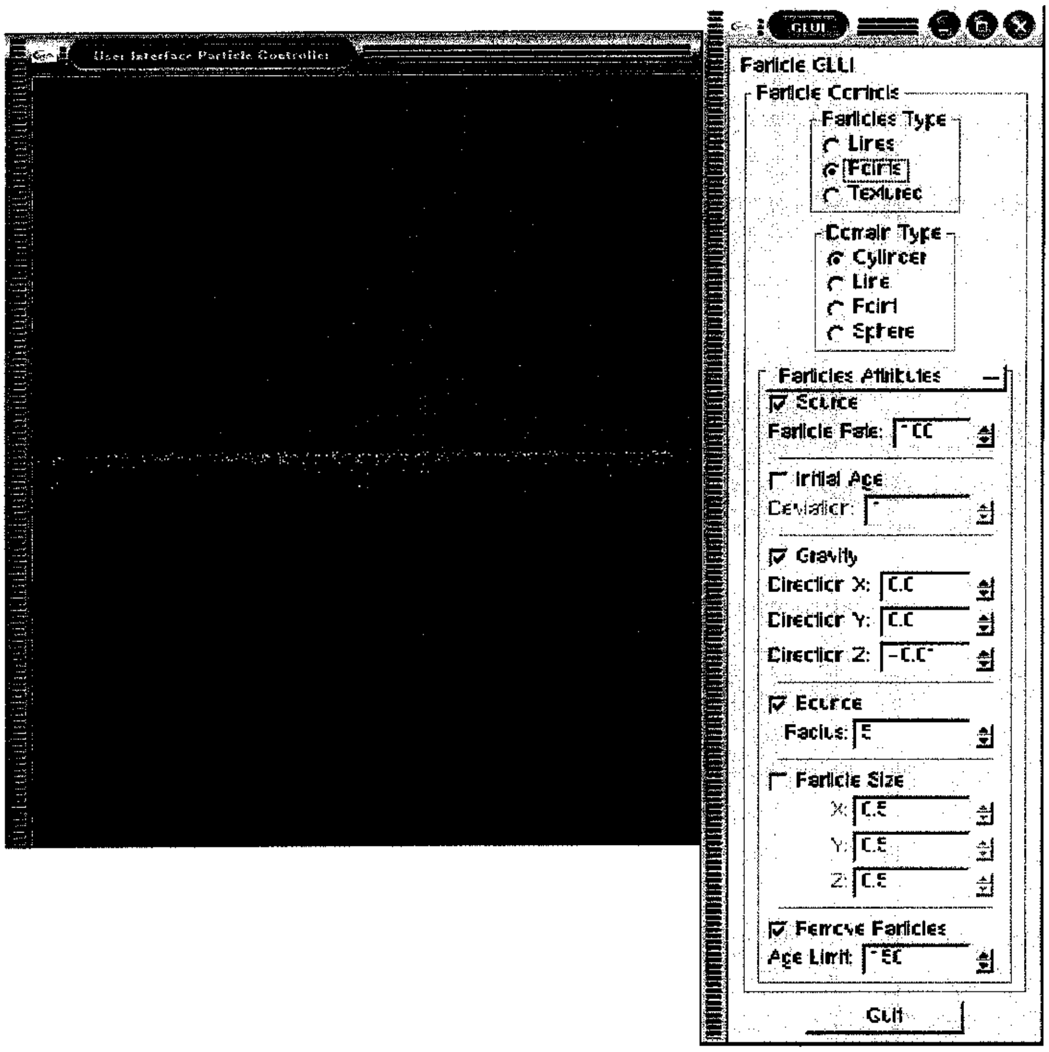
NURUNNISA ABDUL AZIZ

ABSTRACT

The primary objective of this project is to develop a particle system application using Particle System Application Programmer Interface (API) and a GLUT Based User Interface (GLUI). Particle System API is a C++ function library specification that allows applications to simulate the dynamics of the particles. The API was implemented in this project to add a diversity of particle-based properties to interactive graphics applications. The particle system specification is emphasized on the creation of a group of particles that are 'manipulatable'.

GLUI User Interface Library provides a standard user interface elements that allows user interface elements to be added within an OpenGL Utility Toolkit (GLUT) and it gives the user an opportunity to control the particle systems. The integration of both Particle System API and GLUI lead to the paradigm shift in learning and developing the particle systems.

SCREENSHOT OF PARTICLE SYSTEM APPLICATION



Particle GLUI

Particle Controls

Particles Type: Lines [✓ Points] Textured

Domain Type: [✓ Cylinder] Line Point Sphere

Particles Attributes

[✓] Source
Particle Rate: 100

[ ] Initial Age
Deviation: 1

[✓] Gravity
Direction X: 0.0
Direction Y: 0.0
Direction Z: -0.01

[✓] Bounce
Radius: 5

[ ] Particle Size
X: 0.5
Y: 0.5
Z: 0.5

[✓] Remove Particles
Age Limit: 150

Figure 1.0: Screenshot from Particle System Application

ACKNOWLEDGEMENT

The author would like to express her gratitude to Allah the Almighty for giving the opportunity to complete this final year project. Also the author wishes to thank Mr. Mohamed Nordin Zakaria, Final Year Project (IT) Supervisor for his guidance and support throughout the length of the project. His advice and ideas have proven invaluable to the author's effort in completing the project.

Thanks also goes to all the final year IT lecturers and friends for their support, constructing ideas and comments in improving and adding values to this project. It is almost impossible to put the list of all their names here. Their contribution will always be remembered either directly or indirectly. Special thanks to the author's friends, Asrol Affendy bin Gzali and Mohd Taqiyuddin Bakir, who even through rough times believed and supported the author in her efforts.

Last but not least, the author wishes to dedicate her work to her beloved parents, family and colleagues who have always been there when needed. All the support given throughout the years were heartening and highly appreciated.

Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background of Study

Ever since the advent of the computer graphic, the field has become more sophisticated and complicated and the demand for realism, quality and interaction increases in both computer generated effects, in video games and other simulations. Game engines such as Quake Engine have become an essential element for abstracting the representation and dynamics of a video game's virtual world.

1.2 Problem Statement

Simulation of computer generated effects has been used in computer animation for several years and has recently been used in real time simulation and video games. A lot of research has been done to explore ways to compute and render the particle systems but somewhat little research has discussed a suitable application that has interface which can control the particle systems. Therefore, little information is available on an issue regarding the development of an application that can synchronize with the changes of the particles' characteristics without recompiling and running the application many times.

Figure 1.2: Particle System Application

1.2.1 Problem Identified

1. A visible lack of user interface for computer graphics applications and simulations.
2. Most of the user interface of the computer graphics application and simulations is developed using quite a complex programming method.
3. Many of current computer generated effects are developed and displayed without user interface controller and any changes of value of the effects must be made in the development software and need to recompile and run a lot of times which is very tedious task.

1.3 Objectives and Scope

The project's aim is to achieve a number of objectives by the end of the specified period of time prearranged for the course of the project:

- Develop a particle system application to be run in Windows platform
- Implement a user interface that can enable users to manipulate with the particle's attributes

Developing a major particle system application title would be a tedious task which requires time and effort that would exceed the constraints of this project. Therefore, the scope has been correspondingly reduced to ensure the feasibility of this project:

- The particle system application will integrate with a graphical user interface to enable the manipulation of the particle systems
- The development of particle system application is based on Particle System API

In essence, the scope of the project entails the review and understanding of the C++ programming language, 3D graphic programming with OpenGL, Particle System API and GLUI library that will be used for the core of the particle system application.

1.4 Significance of the Project

The particle system application will benefit and provide new alternatives especially to beginners, multimedia and 3D graphics development in which they will be able to understand particle's behaviour and manipulate with the particle system application.

It is hoped that with the completion of the project, the author will have grasped a basic understanding of the principles of graphic development. Readers of this paper should also hopefully achieve a somewhat familiar appreciation of developing particles and the world of graphic development in general.

In both cases, this paper should encourage computer graphic enthusiasts and computer graphic programmers to try and experiment with the particle system development and in the long run, produce worthy and realistic particle simulation.

CHAPTER 2
LITERATURE REVIEW

There are several references that have been done related to the topic. Most of the references are taken from research institutes and paperwork from other universities such as from Department of Computer Science University of North Carolina at Chapel Hill and other institutes and articles.

2.1 Particle System

The idea of using particle system application was first brought to fame by William T. Reeves (Lucasfilm, Ltd.) in a paper called "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects". In the paper, he describes the basic model of a particle system, describes how particle systems differ from other methods of modelling, and describes some potential applications of particle systems.

According to Reeves (1983)

> The representation of particle systems application differs in three basic ways from representations normally used in image synthesis. First, an object is represented not by a set of primitive surface elements. Second, a particle system is not a static entity. Third, an object represented by a particle system is not deterministic, since its shape and form are not completely specified. Instead, stochastic processes are used to create and change an object's shape and appearance.

Reeves (1983) gives us a few characteristics or attributes that need to be determined in developing particle system application:

- Position
- Velocity (speed and direction)
- Size

- Colour
- Transparency
- Shape
- Gravity

Reeves (1983) mentioned one of the advantages of particle systems over other methods which is particles are simple. Because of this, it is possible to render more particles, resulting in the ability to render more complex images. Particle systems are dynamic by their very nature which means that they are naturally suited to animation.

To calculate each frame in a motion sequence, the following steps are performed:

1. New particles are generated into the system
2. Each particle is assigned its individual attributes
3. The prescribed lifetime of the existed particles are extinguished
4. The particles are moved and transformed according to their dynamic attributes
5. An image of the living particles is rendered in a frame buffer

The particle system can be programmed to perform any set of instructions at each step. It is a procedural approach and it can incorporate to any computational model that describes the appearance of the object.

2.2 Particle System API

Particle System Application Programmer Interface (API) was developed by David K. McAllister (University of North Carolina). In a paper called "Documentation for the Particle System API", he portrays particle system API as a set of functions that allow C++ programs to simulate dynamics of particles.

Particle System API is aimed for special effects in graphic applications. It also proposed to be similar to OpenGL from Silicon Graphics, Inc. (SGI). McAllister (1999) found that a particle within the Particle System API is an object with a small set of attributes which is very similar to Reeves' (1983) original particle systems. The particles can be operated on many similar objects that move according to the same rules.

The purpose of the API is to enable real-time applications in developing particle system application. Thus, it requires an efficiency computational method of the particles so the CPU has enough time per frame to perform the application's computation. The API also allows user to design or create effects that are not visualized by other developers.

The API also consists of simple coding that can guide author to understand the API as an overview. Generally there are four (4) set of functions included in the API:

1. Actions

   Actions are functions in API that manipulate attributes of particles in the particle groups. Actions simulate effects or physical forces such as gravity, bouncing, explosions and etc. The API has twenty seven (27) action functions and each particle effects consist of between three to eight actions. Each action performed will be distributed over all particles in the group. pDrawGroupp is used to render particle group whereby each particle being a primitive and for each particle that used display list will call pDrawGroupl.

2. Particle Groups

   Particle group is a collection of particles and each particle exist within a particle group that acted together. All actions apply to every particle in the particle group. It is created using pGenParticleGroups which is to generate the particle group. Maximum number of particles in the group is specified using pSetMaxParticles. When particle group reaches the maximum size, the addition particles will be ignored.

3. Action Lists

   Actions that are compiled into action lists will encapsulate all operations required to produce particular effect. It allows specific effect to be treated as primitive like actions and allows effects to be simulated efficiently. Action lists are produced using pGenActionLists followed by pNewActionList.

   The concept is quite similar with display list of OpenGL. By using the syntax, all subsequent action and state change calls are stored in the action list instead of being executed at once. The pEndActionList will end the list and the API will be switched to normal execute mode. To call the function within the action list, pCallActionList is executed.

   Using action list can reduce the interaction between the applications and the hardware devices. Thus, it improves the application performance.

4. Attributes and Domains

   pSource is one of the API's important actions. pSource is used to create new particles. The particles must be given an attributes such as colour, velocity, size and initial age. In order to increase the flexibility of the API, those attributes are created as API state. Domains have variety of shapes. Domains define 3D volume such as PDSphere, PDPlane, PDBox, PDCylinder and so on.

   Domains also provide a consistent method for identifying API region. For instance, pColorD specifies a region of colour space for new particles and pVelocityD specifies a region of vector space in choosing the velocity of each particle. Domains are used as parameters to some actions and functions. pSource creates particles with position chosen within a domain and so does pSink that kills particles that enter or leave a domain.

2.3 GLUT-Based User Interface Library (GLUI)

GLUI was created by Paul Rademacher (1999). In a document called "GLUI Manual", he explains that GLUI is a GLUT-based C++ user interface library which provides controls such as buttons, checkboxes, radio buttons, spinners, and listboxes to OpenGL applications. It is window-system independent that relies on GLUT to handle all mouse and window management.

The OpenGL Utility Toolkit (GLUT) is a well known user interface library for OpenGL applications which provides a simple interface for handling windows, keyboard, mouse and other input devices. It offers an attractive environment for developing cross platform graphics applications. A lot of features can be used in GLUT and it is common for GLUT applications where almost key on the keyboard is assigned to some functions.

According to Rademacher (1999), GLUI User Interface Library can address with the problem by providing standard user interface elements such as buttons and checkboxes. The GLUI library is written entirely over GLUT and has no system-dependent code. A GLUI program will behave the same on SGIs, Windows machines, Macs, or any other system to which GLUT has been ported. Furthermore, GLUI is designed for programming simplicity, allowing user interface elements to be added with one line of code each.

Rademacher (1999) insists that GLUI is built on top of and meant to fully interact with the GLUT toolkit. Existing GLUT applications therefore need very little change in order to use the user interface library.

2.3.1 Usage for Standalone Windows

Rademacher (1999) found that integrating GLUI with a new or existing GLUT application is very straightforward. The steps are:

1. Add the GLUI library to the link line (glui32.lib). The proper order in which to add libraries is: GLUI, GLUT, GLU and OpenGL.

2. #include the file "glui.h" in all sources that will use the GLUI library.

3. Create regular GLUT windows and popup menus as usual. Make sure to store the window id of main graphics window, so GLUI windows can later send it redisplay events:

   ```
   int window_id = glutCreateWindow("Main gfx window");
   ```

4. Register GLUT callbacks as usual.

5. Register GLUT idle callbacks if any with GLUI_Master (a global object which is already declared), to enable GLUI windows to take advantage of idle events without interfering with the application's idle events. If do not have an idle callback, pass in NULL.

   ```
   GLUI_Master.set_glutIdleFunc(myGlutIdle);
   ```

   or

   ```
   GLUI_Master.set_glutIdleFunc(NULL);
   ```

6. In idle callback, explicitly set the current GLUT window before rendering or posting a redisplay event. Otherwise the redisplay may accidentally be sent to a GLUI window.

```
void myGlutIdle(void)
{
        glutSetWindow(main_window);
        glutPostRedisplay();
}
```

7. Create a new GLUI using

```
GLUI *glui = GLUI_Master.create_glui("name", flags, x, y);
```

Note that flags, x and y are optional arguments. If they are not specified, default values will be used. GLUI provides default values for arguments whenever possible.

8. Add controls to the GLUI window. For example, add a checkbox and a quit button with:

```
glui->add_checkbox("Lighting", &lighting);
glui->add_button ("Quit", QUIT_ID, callback_func);
```

9. Let each GLUI window created know where its main graphics window is:

```
glui->set_main_gfx_window(window_id);
```

10. Invoke the standard GLUT main event loop, just as in any GLUT application:

```
glutMainLoop();
```

CHAPTER 3
METHODOLOGY/PROJECT WORK

For the development of the particle system application, an exploratory research will be used. Exploratory research will rely on secondary sources such as reviewing available literature on journals and articles. Through thorough research, the particle system application then will be developed and experimented based on the findings on Particle System API and GLUI in order to integrate it.

The following is a brief description of the steps taken in the development of the project.

3.1 Project Development Phases

The author's objective is to accomplish the exploratory research phase of this project:

3.1.1 Preliminary Understanding

Preliminary understanding is obtained by expanding research by reviewing previous works or previous articles available to gain familiarity with the project that can lead to narrow down the research study to developing the problem statement.

3.1.2 Literature Search

This phase involves gathering information from secondary sources such as literature review. It is a review of books as well as articles in journals or professional literature is to find ways that address to a solution or solved problems that relates to the author's problem statements. It also involves the analyzing of the requirements for the project in order to obtain the specifications for the project. The analysis of the requirements will result in an initial product specification that the end product will revolve upon.

3.1.3 Data Gathering

Through the literature search, the background information that related to the particle system application is gathered. From the information gathering, the author can identify information that should be gathered and identify the sources for the topic that might be used in the development of the project.

3.1.4 Development

This stage is not really a part of the Exploratory Research, but the author decides to include this stage. This stage is where actual coding for the development of the initial application is carried out. An initial application based on the information gathered and analyzed earlier is used as the constraints to model this project.

3.1.5 Testing and Debugging

Testing and debugging are done to ensure the workability and the functionality of the project.

3.1.6 Final Release

The final product is obtained after all the steps of the exploratory method is complete and a fully functional product is developed meeting the problem statements criteria.

3.2 Tools Required

3.2.1 Hardware

Mentioned below is the minimum requirement that required to develop the application.

- Pentium III, 800 Mhz Processor or higher
- 128 MB RAM or higher
- 3 GB of hard disk space or higher
- 800 X 600, 256 colors of video resolutions, or higher
- Graphics Card – Gforce 64MB or higher

3.2.2 Software

- Visual Studio.NET

  Visual Studio.NET is used in developing the particle system application by using a console Win32. The particle system application will have two windows displayed which are a console and the OpenGL Utility Toolkit (GLUT) window.

CHAPTER 4
RESULTS AND DISCUSSIONS

This section discusses the results that have been achieved upon the implementation of Particle System API and GLUI libraries used in the development of the user interface for particle system controller. Basically, the result will be the end product, which is essentially a particle system application that is 'manipulatable'. The discussions will include the problem encountered throughout the development process.

4.1 The Particle Environment

Based on the literature review done for the purpose of this project, the author started developing the framework for the application using OpenGL and implemented Particle System API and GLUI libraries to build the particle application with user interface controller. The extra headers that are required to be included in the source files are tga.h, papi.h and GL/glui.h. The first header file allows the texturing on the quads whereas papi.h is referred to particle API which provides initialization of the particles. The glui.h allows for the creation of user interface of the particle system application.

The particle system application is developed using win32 console application in Visual Studio.NET. A 500 by 500 pixel window was created as the main window of the particle system application with an additional 220 by 600 pixel sized window at the right side of the main window acting as the particle controller which enable user to manipulate with the particles' attributes.

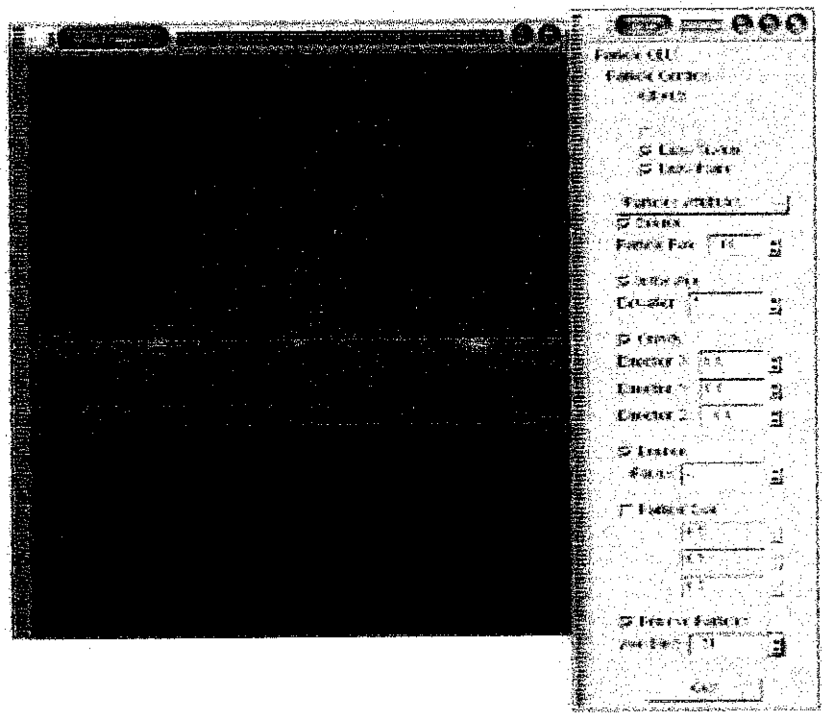


Figure 4.1: The environment of the particle systems and the particle controller.

Initially, the GLUT window will display a plane and user has to make a start by using the particle controller at the right side to choose the particles type. Particles type consists of primitives and textured particles drawn as a group of particles. Primitives are the fastest OpenGL based method of drawing particles. Primitives are point, triangles and lines drawn as a group of particles. When primitive equals points, each particle becomes a single vertex. For lines, each particle becomes a line specified by two vertices, yielding a line in the direction that the particle is moving. A textured particle is drawn by loading the texture, the image in .tga format. The texture then has to be blend using glBlendFunc( ) in order to allow the transparency of the particles.

Domain type is to describe the velocities of the particle. Particles attributes which is using rollout contains the characteristics of the particles to be manipulated by user. User can set their own particle behaviour by using the controller. Source will enable user to add particles in the specified domains. A domain used in source is to describe the volume in which a particle will be created. Target colour contains three (3) primary colour which are red, green and blue. It changes the particle colour towards the specified colour. The gravity section consists of direction x, y and z which is to enable user to indicate and accelerate the particles in the given directions. The bounce part is meant to bounce particles off a domain space. Bounce actions use domains to describe volume in the environment for particles to bounce off respectively. The particle size which consists of size x, y and z is to identify the new size of the particles. Lastly, the remove particle section is meant for removing or kills old particles depending on the value of the age limit set in the spinner.

4.2 Future Upgrades and Recommendations

The current version of particle system controller is far from perfect. Quite a number of areas could be tweaked and codes restructured to provide more functionality and realistic. Below are listed some of the possible recommendations that the author has identified for future work on the project:

- The current version only created particle system application. Future versions might include a particle system application that can simulate effects such as smoke, explosions, fire, rain and other effects.
- Provide a controller that can change the behaviour of the particles so users can manipulate and learn more about the particle system.
- Provide a virtual environment that can simulate how the particle works.

CHAPTER 5
CONCLUSION

Over a period of time, particles are generated into the system; move and change form within the system, and die from the system. As a whole, Particle System models an object to represent motion, changes of form, and dynamic which is not possible with traditional surface-based representations. The implementation of particle system in this project can lead to new experience of virtual environment that represents real world.

In summary, the project has shown significantly that the traditional way of simulating an effect such as controlling the particles using a keyboard can be developed into an interactive way by implementing both Particle System API and GLUI user interface into the coding. Particles are made easy with Particle System API. Particle System and GLUI library is an efficient means of the development of the particles without having to worry too much about how the particles might behave and the behaviour of the particles can be understand by manipulating the particles attributes using GLUI user interface.

The extensive usage of Particle System API shows the advantages of using it. Applying Particle System API helps in managing the particles in group and reduces the complexity of the project.

REFERENCES

Reeves, William T., 1983, *Particle Systems: A Technique for Modeling a Class of Fuzzy Objects*

David, K. McAllister, 1999, *Documentation for the Particle System API*, University of Carolina

Rademacher, Paul, 1999, *GLUI Manual*

David, K. McAllister, 2000, *The Design of an API for Particle Systems*, University of Carolina

J. L. Neider, T. R. Davis and M. Woo, 1993, *OpenGL Programming Guide: The Official Guide to Learning OpenGL*, Addison-Wesley

Wright, Richard S. and Sweet, Michael, 2000, *OpenGL: Super Bible*, Waite Group Press

Sekaran, Uma, 2003, *Research Methods for Business: A Skill Building Approach*, John Wiley & Sons Inc.

Donald, R. Cooper and Pamela, S. Schindler, 2002, *Business Research Methods*, Mc Graw Hill

APPENDIX

SOURCE CODE FOR PARTICLE SYSTEM API

```
//*******************************************************************
      //
      // File Name: particleglui.cpp
      // Author    : Nurunnisa Abdul Aziz
      // ID        : 3089
      // Description: Particle Controller using GLUI and
      //                       Particle API
      //
//*******************************************************************

// link the lib files to the program
#pragma comment(lib, "opengl32.lib")
#pragma comment(lib, "glut32.lib")
#pragma comment(lib, "glui32.lib")
#pragma comment(lib, "particleDLL.lib")

#include <stdlib.h>
#include <GL/glut.h>

// Note that in order to be able to compile a glui application
// it is a must to have glui installed on the computer, and
// glui header file must be included in the folder GL in the
// VC++ folder. Be careful, the file must be in C++ because
// glui doesn't work in C.
// glui32.lib file should be put into the lib direcotry.
#include <GL/glui.h>

// Include particle API header file in all applications that use
// the Particle System API.
#include "papi.h"

// Include tga loader header file in order to load .tga file.
#include "tga.h"

// Include a camera header file to set up camera implementation
// in the application.
#include "CCamera.h"

// Check the input for each frame.
void GetInput();
// Camera object used to move around the scene.
CCamera Camera;

// define the display list
GLuint theQuad;

// declaration for texture
GLuint LoadTexture(char *TexName);
// texture ID
GLuint TexID1;
```

```
// These are live variables that will be passed into glui.
int   main_window;

// define primitives properties
int   particles = 0;
int   particles_type = 0;

// define domain type of the particles
int domain = 0;
int domain_type = 0;

// define target color consists of red, green, blue
int targetC = 0;
int r = 0;
int g = 0;
int b = 0;

// define remove particle properties
int kill = 1;
int age_limit = 10;

// define size properties
int   size = 0;
float size_x = 0.5;
float size_y = 0.5;
float size_z = 0.5;

// define gravity properties
int      gravity = 0;
float dir_x = 0.0;
float dir_y = 0.0;
float dir_z = 0.0;

// define source properties
int   source = 1;
int   particle_rate = 10;

// define bounce properties
int      bounce = 0;
int   radius = 0;

/* GLUI CODE */
// Pointers to the windows and some of the controls created
GLUI              *glui;
GLUI_RadioGroup   *radio1, *radio2;
GLUI_Checkbox     *checkbox1, *checkbox2, *checkbox3, *g_checkbox,
                  *b_checkbox, *size_checkbox, *checkbox,
                  *color_checkbox;
GLUI_Panel        *obj_panel;
GLUI_Rollout      *options;
GLUI_Spinner      *grav_spinner1, *grav_spinner2, *grav_spinner3,
                  *b_spinner, *size_spinner1, *size_spinner2,
                  *size_spinner3, *source_spinner, *target_spinner1,
                  *target_spinner2,*target_spinner3, *kill_spinner;
```

```
// GLUI control callback
void control_cb( int control )
{
        /* this function is totally unnecessary here since the values
         * needed by the code are the same as set by the radiobuttons */

  // If control = particle type's radio button
  if ( control == 1 ) {
    particles = particles_type;

        // If particle type = lines
        // enable target color properties
        if(particles_type == 0)
        {
                color_checkbox->enable();
                target_spinner1->enable();
                target_spinner2->enable();
                target_spinner3->enable();
        }

        // If particle type = points
        // enable target color properties
        else if(particles_type == 1)
        {
                color_checkbox->enable();
                target_spinner1->enable();
                target_spinner2->enable();
                target_spinner3->enable();
        }

        // If particle type = textured
        // disable target color properties
        else if(particles_type == 2)
        {
                size_checkbox->enable();
                color_checkbox->disable();
                target_spinner1->disable();
                target_spinner2->disable();
                target_spinner3->disable();
        }
  }
  else if ( control == 16)
  {
        if(source)
        {
                source_spinner->enable();
        }
        else
        {
                source_spinner->disable();
        }
  }
```

```
else if ( control == 18)
{
    if(targetC)
    {
            // Enable target color properties
            target_spinner1->enable();
            target_spinner2->enable();
            target_spinner3->enable();
    }
    else
    {
            // Disable target color properties
            target_spinner1->disable();
            target_spinner2->disable();
            target_spinner3->disable();
    }
}
else if ( control == 5)
{
    if(gravity)
    {
            // Enable gravity properties
            grav_spinner1->enable();
            grav_spinner2->enable();
            grav_spinner3->enable();
    }
    else
    {
            // Disable gravity properties
            grav_spinner1->disable();
            grav_spinner2->disable();
            grav_spinner3->disable();
    }
}
else if ( control == 9)
{
    if(bounce)
    {
            // Enable bounce properties
            b_spinner->enable();
    }
    else
    {
            // Disable bounce properties
            b_spinner->disable();
    }
}
else if ( control == 12)
{
    if(size)
    {
            // Enable size properties
            size_spinner1->enable();
            size_spinner2->enable();
            size_spinner3->enable();
    }
```

```
        else
        {
                // Disable size properties
                size_spinner1->disable();
                size_spinner2->disable();
                size_spinner3->disable();
        }
    }
    else if ( control == 20)
    {
        if(kill)
        {
                // Enable remove particle properties
                kill_spinner->enable();
        }
        else
        {
                // Disable remove particle properties
                kill_spinner->disable();
        }
    }
}

// Particles spraying up in the middle of the screen
void ComputeParticles()
{
        // Set up the state function.
        // Target velocity of particles
        if(targetC){
                pTargetColor(r, g, b, 1.0, 0.5);
        }

        // describe the particles' velocities
        // if domain type = cylinder
        if(domain_type == 0){
                pVelocityD(PDCylinder, 0.01, 0.0, 0.035, 0.01, 0.0, 0.37,
                        0.021, 0.019);
        }

        // If domain type = line
        else if(domain_type == 1){
                pVelocityD(PDLine, 0.01, 0.0, 0.035, 0.01, 0.0, 0.37,
                        0.021, 0.019);
        }

        // If domain type = point
        else if(domain_type == 2){
                pVelocityD(PDPoint, 0.01, 0.0, 0.35, 0.01, 0.0, 0.37,
                        0.021, 0.0019);
        }

        // If domain type = sphere
        else if(domain_type == 3){
                pVelocityD(PDSphere, 0.0, 0.0, 0.0, 0.5, 0.0);
        }
        // initial age of particles (float age&stdev)
        pStartingAge(1.0, 2.0);

        // Size of the particles.
        if(size){
                pSize(size_x, size_y, size_z);
        }

        // Generate particles along a very small line in the nozzle.
        if(source){
                pSource(particle_rate, PDLine, 0.0, 0.0, 0.0, 0.0, 0.0,
                0.0);
        }

        // Gravity.
        if(gravity){
                pGravity(dir_x, dir_y, dir_z);
        }

        //remove old particles that older than age limit.
        if(kill){
                pKillOld(age_limit, false);
        }

        // Bounce particles off a disc of radius.
        if(bounce){
                pBounce(-0.05, 0.35, 0, PDDisc, 0, 0, 0,  0, 0, 1,
                radius);
        }

        // Kill particles below Z=-3.
        pSink(false, PDPlane, 0,0,-3, 0,0,1);

        // Move particles to their new positions.
        pMove();

}

static void init(void)
{
        // Load the textures
        TexID1=LoadTexture("sprite.tga");

        // Enable blending
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE);

        // begin display list
        theQuad = glGenLists(1);
        glNewList(theQuad, GL_COMPILE);

        glBegin(GL_QUADS);
                glColor3f(0.8f, 0.1f, 0.2f);  // adjust particle color
                glTexCoord2f(1.0f,1.0f); glVertex3f(0.0f, 0.0f, 0.0f);
                glTexCoord2f(1.0f,0.0f); glVertex3f(0.1f, 0.0f, 0.0f);
                glTexCoord2f(0.0f,0.0f); glVertex3f(0.1f, 0.0f, 0.1f);
                glTexCoord2f(0.0f,1.0f); glVertex3f(0.0f, 0.0f, 0.1f);
```

```
      glEnd();

      glEndList();

      glClearColor(0.0, 0.0, 0.0, 0.0);

      // Here we set the camera's default position.
      Camera.SetCamera(0.0f, 0.2f, -10.0f, 0.0f, 0.0f, 0.0f, 0.0f,
 1.0f, 0.0f);
}

// Load a TGA texture
GLuint LoadTexture(char *TexName)
{
      TGAImg Img;          // Image loader
      GLuint Texture;

      if(Img.Load(TexName)!=IMG_OK)
            return -1;

      // Allocate space for texture
      glGenTextures(1,&Texture);
      // Set our Tex handle as current
      glBindTexture(GL_TEXTURE_2D,Texture);

            // Create the texture
            if(Img.GetBPP()==24)

      glTexImage2D(GL_TEXTURE_2D,0,3,Img.GetWidth(),Img.GetHeight(),0,
GL_RGB,GL_UNSIGNED_BYTE,Img.GetImg());
            else if(Img.GetBPP()==32)

      glTexImage2D(GL_TEXTURE_2D,0,4,Img.GetWidth(),Img.GetHeight(),0,
GL_RGBA,GL_UNSIGNED_BYTE,Img.GetImg());
            else
            return -1;

      // Specify filtering and edge actions
      glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
      glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
      glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
      glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);

      return Texture;
}

void display(void)
{
/* display callback, clear frame buffer and z buffer,
   and draw, swap buffers */

      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glLoadIdentity();

      // The get input function will check for input for each frame.
      GetInput();
```

```
    // gluLookAt() will position the camera.
    gluLookAt(Camera.Pos.x, Camera.Pos.y, Camera.Pos.z,
              Camera.View.x, Camera.View.y, Camera.View.z,
              Camera.Up.x, Camera.Up.y, Camera.Up.z);

    glScalef(2.0, 2.0, 2.0);

    // Position our object.
    // Our 3rd person camera will be following this.
    glTranslatef(Camera.View.x, 0, Camera.View.z);

    glRotatef(180, 0.0, 1.0, 0.0);

    // Draw the ground.
    glBegin(GL_QUADS);
    glColor3ub(0, 115, 0);
    glVertex3f(-5.0, 0.0, -5.0);
    glColor3ub(0, 5, 140);
    glVertex3f(-5.0, 0.0, 5.0);
    glColor3ub(0, 5, 140);
    glVertex3f(5.0, 0.0, 5.0);
    glColor3ub(0, 115, 0);
    glVertex3f(5.0, 0.0, -5.0);
    glEnd();

    // Do what the particles do.
    ComputeParticles();

    if(particles_type == 0){

          glRotatef(-90, 1.0, 0.0, 0.0);
          pDrawGroupp(GL_LINES, true);
    }

    else if(particles_type == 1){

          glRotatef(-90, 1.0, 0.0, 0.0);
          pDrawGroupp(GL_POINTS, true);
    }

    else if(particles_type == 2){
          glEnable(GL_TEXTURE_2D);
          glBindTexture(GL_TEXTURE_2D,TexID1);

          glRotatef(-90, 1.0, 0.0, 0.0);

    //    pDrawGroupl(dlist, size, color, rotation)
          pDrawGroupl(theQuad, true, true, true);

          // Disable the texture.
          glDisable(GL_TEXTURE_2D);
    }

    glutSwapBuffers();
}
```

```
void myIdle()
{
	/* GLUI CODE */
	// According to the GLUI specification, the current
	// window is undefined during an idle callback. So
	// we need to explicitly change it if necessary.
	  if ( glutGetWindow() != main_window )
		glutSetWindow(main_window);

	glutPostRedisplay();

  /*****************************************************************/
  /*              This demonstrates GLUI::sync_live()             */
  /*   The value of a variable is changed that is 'live' to some  */
  /*   control.  Then sync_live is called, and the control        */
  /*   associated with that variable is automatically updated     */
  /*   with the new value.                                        */
  /*****************************************************************/
	glui->sync_live();
	radio1->set_int_val(particles);
}

void mouse(int btn, int state, int x, int y)
{

/* mouse callback, selects particles */

	if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) particles = 0;
	if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) particles = 1;
	if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) particles = 2;

}

void myReshape(int w, int h)
{
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);

	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(60, w / double(h), 1, 100);
	glMatrixMode(GL_MODELVIEW);
}

void GetInput()
{
	// Update the time for time based movement.
	Camera.CalculateTime();

	// Here we check for each input that we care about.
	if(GetKeyState(VK_UP) & 0x80) Camera.MoveCamera(UP);
	if(GetKeyState(VK_DOWN) & 0x80) Camera.MoveCamera(DOWN);

	// Rotate the camera left in 3rd person.
	if(GetKeyState(VK_LEFT) & 0x80) Camera.RotateCamera
	(CVector4(Camera.View.x, Camera.View.y, Camera.View.z),
	LEFT, 0, 1, 0);
```

```
        // Rotate the camera right in 3rd person.
        if(GetKeyState(VK_RIGHT) & 0x80)
        Camera.RotateCamera(CVector4(Camera.View.x, Camera.View.y,
        Camera.View.z), RIGHT, 0, 1, 0);

        // Rotate the camera up in 3rd person.
        if(GetKeyState('W') & 0x80 || GetKeyState('w') & 0x80)
        Camera.RotateCamera(CVector4(Camera.View.x, Camera.View.y,
        Camera.View.z), UP, 1, 0, 0);

        // Rotate the camera down in 3rd person.
        if(GetKeyState('S') & 0x80 || GetKeyState('s') & 0x80)
        Camera.RotateCamera(CVector4(Camera.View.x, Camera.View.y,
        Camera.View.z), DOWN, 1, 0, 0);
}

void main(int argc, char* argv[])
{
  /****************************************/
  /*   Initialize GLUT and create window  */
  /****************************************/

  glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
  glutInitWindowPosition( 50, 50 );
  glutInitWindowSize( 500, 500 );

  main_window = glutCreateWindow( "User Interface Particle Controller"
);
  glutDisplayFunc( display );
  glutReshapeFunc( myReshape );
  glutMouseFunc(mouse);

  glEnable(GL_DEPTH_TEST);
  glDepthFunc(GL_LESS);

  init();

  // Make a particle group
  int particle_handle = pGenParticleGroups(1, 4000);
  pCurrentGroup(particle_handle);

  /****************************************/
  /*         Here's the GLUI code         */
  /****************************************/

  // to print glui version
  printf("GLUI version: %3.2f \n", GLUI_Master.get_version());

  // Now create a GLUI user interface window and add controls
  // name, flags, x, and y
  glui = GLUI_Master.create_glui( "GLUI", 0, 400, 50 );
  glui->add_statictext( "Particle GLUI" );
  obj_panel = glui->add_panel( "Particle Controls" );
  // radiobutton - handling options
  // Particles Type
  GLUI_Panel *type_panel1 = glui->add_panel_to_panel( obj_panel,
"Particles Type");
  radio1 = glui->add_radiogroup_to_panel(type_panel1, &particles_type,
1, control_cb);

  glui->add_radiobutton_to_group(radio1, "Lines");
  glui->add_radiobutton_to_group(radio1, "Points");
  glui->add_radiobutton_to_group(radio1, "Textured");

  // Domain type
  GLUI_Panel *type_panel2 = glui->add_panel_to_panel( obj_panel,
"Domain Type");
  radio2 = glui->add_radiogroup_to_panel(type_panel2, &domain_type, 2,
control_cb);

  glui->add_radiobutton_to_group(radio2, "Cylinder");
  glui->add_radiobutton_to_group(radio2, "Line");
  glui->add_radiobutton_to_group(radio2, "Point");
  glui->add_radiobutton_to_group(radio2, "Sphere");

  // Rollout - to group controls into collapsible boxes
  options = glui->add_rollout_to_panel(obj_panel, "Particles
Attributes", false);

  // Checkboxes - handling booleans
  checkbox = glui->add_checkbox_to_panel(options, "Source", &source,
16, control_cb);

  // Spinner - interactively manipulating numeric values and supports
clicks
  // Source properties
  source_spinner = glui->add_spinner_to_panel(options, "Number of
Particles: ", GLUI_SPINNER_INT, &particle_rate, 17, control_cb);
  source_spinner->set_int_limits(0, 1000);

  // Separator - separating controls with simple horizontal lines
  glui->add_separator_to_panel(options);

  // Target velocity properties
  color_checkbox = glui->add_checkbox_to_panel(options, "Target
Color", &targetC, 18, control_cb);
  target_spinner1 = glui->add_spinner_to_panel(options, "Red: ",
GLUI_SPINNER_FLOAT, &r, 22, control_cb);
  target_spinner1->set_float_limits(0.0, 1.0);
  // Disable this initially
  target_spinner1->disable();

  target_spinner2 = glui->add_spinner_to_panel(options, "Green: ",
GLUI_SPINNER_FLOAT, &g, 23, control_cb);
  target_spinner2->set_float_limits(0.0, 1.0);
  // Disable this initially
  target_spinner2->disable();
  target_spinner3 = glui->add_spinner_to_panel(options, "Blue: ",
GLUI_SPINNER_FLOAT, &b, 24, control_cb);
  target_spinner3->set_float_limits(0.0, 1.0);
  // Disable this initially
  target_spinner3->disable();
  glui->add_separator_to_panel(options);

  // Gravity controls
  g_checkbox = glui->add_checkbox_to_panel(options, "Gravity",
&gravity, 5, control_cb);
  grav_spinner1 = glui->add_spinner_to_panel(options, "Direction X: ",
GLUI_SPINNER_FLOAT, &dir_x, 6, control_cb);
  grav_spinner1->set_float_limits(-0.1f, 0.1f);
  grav_spinner1->set_speed(0.1f);
  // Disable this initially
  grav_spinner1->disable();

  grav_spinner2 = glui->add_spinner_to_panel(options, "Direction Y: ",
GLUI_SPINNER_FLOAT, &dir_y, 7, control_cb);
  grav_spinner2->set_float_limits(-0.1f, 0.1f);
  grav_spinner2->set_speed(0.1f);
  // Disable this initially
  grav_spinner2->disable();

  grav_spinner3 = glui->add_spinner_to_panel(options, "Direction Z: ",
GLUI_SPINNER_FLOAT, &dir_z, 8, control_cb);
  grav_spinner3->set_float_limits(-0.01f, 0.1f);
  grav_spinner3->set_speed(0.1f);
  // Disable this initially
  grav_spinner3->disable();
  glui->add_separator_to_panel(options);

  // Bounce controls
  b_checkbox = glui->add_checkbox_to_panel(options, "Bounce", &bounce,
9, control_cb);
  b_spinner = glui->add_spinner_to_panel(options, "Radius:",
GLUI_SPINNER_INT, &radius, 11, control_cb);
  b_spinner->set_int_limits(0, 10);
  // Disable this initially
  b_spinner->disable();
  glui->add_separator_to_panel(options);

  // size
  size_checkbox = glui->add_checkbox_to_panel(options, "Particle
Size", &size, 12,control_cb);
  size_checkbox->disable();
  size_spinner1 = glui->add_spinner_to_panel(options, "X:",
GLUI_SPINNER_FLOAT, &size_x, 13, control_cb);
  size_spinner1->set_float_limits(0.0f, 1.5f);
  size_spinner1->set_speed(0.1f);
  // Disable this initially
  size_spinner1->disable();
  size_spinner2 = glui->add_spinner_to_panel(options, "Y:",
GLUI_SPINNER_FLOAT, &size_y, 14, control_cb);
  size_spinner2->set_float_limits(0.0f, 1.5f);
  size_spinner2->set_speed(0.1f);
  // Disable this initially
  size_spinner2->disable();

  size_spinner3 = glui->add_spinner_to_panel(options, "Z:",
GLUI_SPINNER_FLOAT, &size_z, 15, control_cb);
  size_spinner3->set_float_limits(0.0f, 1.5f);
  size_spinner3->set_speed(0.1f);
  // Disable this initially
  size_spinner3->disable();
  glui->add_separator_to_panel(options);

  // kill particles
  checkbox = glui->add_checkbox_to_panel(options, "Remove Particles",
&kill, 20, control_cb);
  kill_spinner = glui->add_spinner_to_panel(options, "Age Limit: ",
GLUI_SPINNER_INT, &age_limit, 21, control_cb);
  kill_spinner->set_int_limits(0, 1200);

  // Button - invoking user actions
  glui->add_button( "Quit", 0,(GLUI_Update_CB)exit );

  // Link windows to GLUI, and register idle callback
  glui->set_main_gfx_window( main_window );

  // Register the idle callback with GLUI, not with GLUT
  GLUI_Master.set_glutIdleFunc( myIdle );

  // Regular GLUT main loop
  glutMainLoop();
}
```