WEST UNIVERSITY OF TIMIŞOARA
FACULTY OF MATHEMATICS AND INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE

SIMPLE API FOR 3D APPLICATIONS

Author:
Lucian F. Ştefănoaica

Scientific Coordinator:
Lector Dr. Marc E. Frîncu

Timişoara, 2015

Contents

Abstract

Computer graphics improves our daily lives by offering computer aided design tools and means of entertainment. Particle systems are what enrich computer graphics by enabling the possibility to render fire, smoke and other interesting natural phenomena using computers. The application programming interface presented in this thesis, alongside implementation related concepts, is actually a small library packed with a couple of graphical effects based on particle systems. The thesis is composed out of four chapters and the content of each chapter is shortly described bellow.

Chapter 1 gives a brief history of computer graphics and particle systems. This chapter also specifies the role that particle systems play in computer graphics and why are they necessary.

Chapter 2 describes some technical details about particle systems. By the end of this chapter a programmer should already have an idea about how to implement such a system in a graphical application.

Chapter 3 presents all the particle based graphical effects in the API and gives implementation details about them. It also gives details about the structure of the API and that of the application which uses the API. Besides this it demonstrates the API's use with a couple of screen-shots.

Chapter 4 discusses some future work directions which should enhance the rendering performance of the API. It also gives a short description of my learning experience.

Abstract

Abstract in Romanian.

Chapter 1

Introduction

1.1 Brief History of Computer Graphics

What is *computer graphics*? The term first appeared in 1960 and it was made-up by William Fetter who was, at the time, a computer graphics researcher for Boeing. The term refers to a subfield of computer science which deals with image data representation and manipulation using computers.

The first computer able to do graphics is the Whirlwind computer. Its development was started in 1945 at MIT by a team of computer scientists led by Jay Forrester. The purpose of this computer was to make aircraft tracking possible on a large oscilloscope screen via a graphical application. Although the aircraft tracking application was the first application in the field of computer graphics it was not interactive. It could only display the real time positions of the tracked aircrafts. The first interactive graphical application was Tennis For Two and it was created by William Higinbotham because he wanted to kill the boredom of the visitors of the Brookhaven National Laboratory.

In 1959 the TX-2 computer emerged. This computer was used by Ivan Sutherland to program the Sketchpad, a tool for creating very precise engineering drawings. The software offered its users the possibility to draw lines and circle arcs. The lines could then be made perfectly parallel or perpendicular in order to fit the users drawing needs. Sketchpad is known to be the first graphical user interface or GUI in short form.

In 1966 Ivan Sutherland made yet another contribution to the field of computer graphics by inventing the Sword of Damocles the first computer

controlled head mounted display. This device displayed two stereoscopic images of the same wire-frame mesh. Two decades later NASA would use his methods in virtual reality research.

Very soon after, in 1970 actually, the field of computer graphics was upgraded by Henri Gouraud, Jim Blinn and Bui Tuong Phong. The first added the Gouraud shading model and the other two added the Blinn-Phong shading model. In 1978 Jim Blinn also added bump mapping to the computer graphics field.

1.2 Brief History of Particle Systems

The term **particle systems** was coined in 1982 by William T. Reeves who was a computer graphics researcher for Lucasfilm Ltd., a film production company known for films like the Star Trek and Star Wars franchises. But what is a particle system? Here's the original definition:

"A particle system is a collection of many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system."

–William T. Reeves–acm Transactions On Graphics–April 1983–Vol. 2, No. 2

At Lucasfilm Ltd. Reeves helped to develop the wall of fire effect which occurred every time the Genesis Device was used in the film Star Trek II: The Wrath of Khan. Though this is not the first time when particle systems have been used in the computer industry, this particular effect helped to coin down the term **particle systems**.

The first particle systems were implemented in the earliest video games where this concept was used to portray exploding spaceships which left behind many little dots or lines. For example "Spacewar!" which appeared in 1962 and "Asteroid" which appeared later in 1978 had such effects implemented. Since their dawn till the very present particle systems have been used to implement various interesting phenomena such as fire, smoke or waterfalls. In the absence of particle systems these phenomena would be very difficult to implement if not impossible.

1.3 The Necessity of Particle Systems

An alternate solution to the problem of modeling objects like stormy clouds, explosions or smoke would be to use surface-based representations (the ones which are used to model objects like cars or spaceships). It is easy to notice though, that it would not be a very natural approach of the problem to try to model an object which has no smooth surfaces, perhaps a cloud of smoke, with a series of triangles where each triangle represents a surface which actually does not exist on the real object. One could easily implement such a smoke cloud using triangles to represent its margins but the results would not be very realistic. If a large number of very small triangles were to be used then the smoke cloud would look better, in fact the smaller the triangles and the larger the count of the triangles the better the quality of the image. But there is a downside to this approach.

Let us imagine what would happen to the previously mentioned smoke cloud if a sudden wind gust were to hit it. The smoke cloud would most certainly move in the direction of the wind. Only with the surface-based representation it would move in the direction of the wind as a whole just like a very large balloon would. The representation of the smoke cloud would not be very realistic because in real life when a smoke cloud gets hit by a lateral wind force some pieces of it go with the wind while others go in completely different directions or move slightly from their initial position. Those pieces of the smoke cloud are actually *particles* of smoke. So if particle systems are used to "implement" smoke clouds in real life, why not use this concept as well, in the implementation of a smoke cloud which would only exist inside a graphical application? Instead of representing only the surface of the smoke cloud using medium to large sized triangles one should represent the whole volume of the smoke cloud using many small sized triangles. This way, not only the visual aspects of the smoke cloud would be improved but also a great amount of flexibility would be added to it. This added flexibility will make the object behave realistically, in other words it will make the smoke cloud react like a real smoke cloud would react not like a balloon which is the case of the surface-based representation. This added flexibility represents the advantage of particle systems over surface-based representations.

1.4 Reasons for Studying Particle Systems

Particle systems represent a fairly important role in computer graphics because without them the problem of rendering natural phenomena like fire, smoke, waterfalls, fog and others would not be solved correctly. There may be other techniques which can be used to treat this problem out there, but one should reckon that using particle systems is the most natural approach to this problem.

I have always been curious of what stands behind those nice fiery graphical effects which appear in most computer games. So naturally, I wanted to learn about them. Programming particle systems represents a good way to learn graphics programming, and not only that. Particle systems are a rather complex concept because their implementation requires notions of mathematics and physics besides the obvious notions of computer science. One of the reasons why I accepted to study this concept is because I wanted to acquire experience in interdisciplinary areas and consolidate my computer graphics knowledge. With this experience I can further improve my professional career by writing graphic applications for both desktop and mobile devices.

Typically most programmers use libraries. Why? Because it is faster. Most problems, large or small, can be divided into subproblems. If one divides a larger problem into a set of ten smaller subproblems he would then have to gradually solve each and every one of the subproblems in order to solve the larger problem. Let us say that every subproblem in the set takes about two to three days to solve, this means that the whole set of subproblems should take, in the worst case, about a month to solve. What if somebody else has already solved half of the subproblems in the set and has packaged the solutions in a library? Then that library would make any programmer who would face the initial larger problem twice as productive because that somebody else has already done half of the work. Java OpenGL (JOGL) is such a library. It offers all of the high level OpenGL function calls necessary to write graphic applications in Java. Although this library is highly comprehensive it does not offer easy ways to implement particle system based graphical effects meaning that, if someone would want to implement a fiery particle system he would have to do so from scratch. So another reason why I accepted to write the simple graphical API presented in this thesis is to extend JOGL. The simple graphical API is completely dependent on JOGL and it offers some simple function calls which will render particle systems of different shapes, sizes, colors or textures.

1.5 Thesis Structure

The thesis is structured in four chapters and the content of each chapter is shortly described bellow.

Chapter 1 gives a brief history of computer graphics and particle systems. After that, this chapter specifies the necessity of particle systems and the role that is played by them in computer graphics. This chapter also presents some reasons why I chose this topic.

Chapter 2 offers technical details about particle systems. It talks about how to implement such systems and how to organize the code in order to achieve the desired results. By the end of this chapter a programmer should already have an idea about how to implement a particle system in a graphic application.

Chapter 3 presents all the particle based graphical effects in the API and gives implementation details about them. It also gives details about the structure of the API and that of the application which uses the API. Besides this it demonstrates the API's use with a couple of screen-shots.

Chapter 4 discusses some future work directions which should enhance the rendering performance of the API. Work directions regarding new effects which can be added to the API later are also mentioned. Besides this it gives a short description of my learning experience.

Chapter 2

Particle Systems

2.1 How To Build Particle Systems

From an object-oriented programming point of view a **particle system** is a simple container object which manages all the particles inside it. This container has an interface which programmers can use to control the system and the particles inside it. But how would one design and implement such a container?

Here are a few steps which can be carried out in order to obtain an object-oriented model of a problem. In order to highlight some of the base components of an object-oriented system a programmer should write down a detailed description of the problem which the system is supposed to solve (here the problem is how to "tell" a computer what a particle system is and how does it behave). After that, the programmer should iterate through the description and identify all the nouns, adjectives and verbs. The nouns represent potential class names, the verbs represent potential class attributes and the verbs represent potential class methods. The next step is to associate class names with class attributes and class methods.

In the case of the particle systems problem, a particle could be a class, and also, the particle system itself could be a class. But these classes should only be generic classes, therefore they should only contain information which is common to all the particles and all the particle systems respectively. This classes will never be instantiated, instead they will serve as parent classes to other concrete classes which will hold specialized information about specific particles and specific particle systems. Some particles may move differently than others and some particle systems may emit more than one type of par-

ticles. But how do particle systems actually work? Every time a particle system is being rendered a few things happen. Being a simple container which manages its contained objects a particle system starts off by being completely empty. At each rendering cycle new particles are created and added to the particle system. The list of contained particles is checked for "dead" particles (these are particles that have exceeded their expected lifetime) in order to remove them from the list. Each particle in the system is first placed in the origin of the system and has its next position calculated with respect to its old position and other attributes like direction, speed or acceleration. After this all the particles in the system are finally being rendered.

2.2 The Particle Spawning Stage

In this stage of the particle systems rendering cycle new particles are created. The particle generation can be done in more than one way. The simplest way to do this is to generate a constant number of particles with each frame that gets rendered on the screen. But in order to improve the obtained results other particle generation methods can be used. These are just some example methods, one could invent quite a lot of such methods.

One method could be to use a mean and a variance counter. The mean represents an average of the number of particles generated throughout a sequence of frames and the variance represents the largest difference between the average and the actual number of particles generated at a specific frame. The following formula shows exactly how the two constants can be used.
n = m + rv where n is the number of particles generated per frame, m is the mean, v is the variance and r is a random number between -1 and 1.

Another method to generate particles could be to take the area of the screen covered by the particle system into consideration as well. In other words if the rendered particle system is very far away from the camera then it will appear to be very small thus the smaller number of particles will not affect the quality of the image. There is no need to use 10,000 particles to render a particle system which takes only one or two squared centimeters of the screen. This is how to achieve this improvement: n = (m + rv)s where n is the number of particles generated, m is the average number of particles generated over a period of time, v is the variance, r is a random number between -1 and 1 and s represents the consumption of screen area.

2.3 Some Basic Particle Attributes

While from a physics point of view a particle can be anything from a photon to a few water molecules or a even star, from an object oriented programming point of view a particle is a simple object which has a state and a behavior. The attributes of a particle represent its state and the methods of a particle represent a way to control its state. A particle can have, more or less, the following attributes: an initial position, an initial speed, an initial acceleration, an initial size, an initial color and a lifetime. The lifetime can be used as a transparency attribute as well. As the lifetime of a particle decreases its transparency increases, thus by the time a particle dies it should already be completely transparent (invisible).

A particle texture attribute can also be added to this basic attributes. Actually, in order to improve the visual aspects of the particle system, a whole list of texture objects can be used. By using this method a particle can be implemented to have a variety of textures throughout its lifetime. The same technique can be used for the shape and color of a particle such that the particle can have a whole range of colors and shapes throughout its lifetime.

The attributes of a particle can be initialized using methods similar to the ones used to compute the number of particles created per frame. For example the initial speed can be computed using the following formula: s = m + rv where s is the computed speed, m is the average of the speed values over a period of time, r is a random number between -1 and 1 and v is the maximum difference between the average speed and the actual speed of a particle. Nearly all the other particle attributes can be initialized using this technique.

2.4 What Makes a Particle Dynamic

Any particle that changes its state (if it moves, if it changes its color, its texture or any attribute whatsoever) is dynamic. A particle which keeps its state constant during its whole lifetime is a static particle. But how to make a particle move or change its color or its transparency?

In order to change a particles position in three dimensional space one must add the speed vector to the old position vector. This is how the new position vector is obtained. This computation is performed with each frame. This is the basic particle movement mechanism. One could add other force vectors (gravity and wind are two good examples) to the position vector to make the movement of the particle more complex.

2.5 When to Kill a Particle

There are a few methods which can be used to determine when to destroy particles from a system. One of these methods is to keep track of a lifetime value which is supposed to drop with each rendered frame. The distance between a particles current position and the origin of the particle system can also be used to determine when a particle leaves a certain region surrounding the origin of the system. If that distance exceeds a certain value, the particle will be removed from the particle system.

With each frame that is rendered on the screen, all the particles from a specific particle system are checked to see if they satisfy their dying condition satisfied. All the particles which satisfy that condition are sequentially removed from the system.

2.6 The Particle Rendering Process

There are more than one methods which can be used to render particles. The particles which formed the Genesis Effect from Star Trek II: The Wrath of Khan implemented by William T. Reeves and a team of computer graphics developers were actually light sources which combined themselves with each other in an additive manner regarding their color and transparency values. This technique removed the hidden surface problem because the particles did not obscure each other but simply added more light to a certain pixel.

A second method would be to use billboards to represent particles. A simple rectangle made out of two simple triangles can be a billboard. In fact any two dimensional element rendered in a there dimensional environment can be a billboard. The main difference between billboards and simple triangles is that a billboard always rotates itself around one or more axes in order to face the camera. There are advantages and disadvantages to using billboards instead of light sources. One advantage is that billboards can be textured with more than one texture over their lifetime. A disadvantage is that billboards may need to be decreasingly sorted according to their distance from the camera. In other words the billboards which are furthest from the camera need to be rendered before the billboards which are closer to the camera otherwise the blending effect will be spoiled.

Chapter 3

Application

This is the application chapter.

Chapter 4

Conclusion

This is the conclusion chapter.