REVIEW OF RESEARCH
IMPACT FACTOR : 5.7631(UIF) UGC APPROVED JOURNAL NO. 48514 ISSN: 2249-894X
VOLUME - 8 | ISSUE - 1 | OCTOBER - 2018

EXPLOSION DEMO

Roopa R. L.
Department of Computer Science and Engineering, PDA College of Engineering, Gulbarga.

ABSTRACT

The project "EXPLOSION DEMO" is an example to show that computer graphics is one of the most interesting and fascinating thing in the computer world. This demo shows how to use an extremely simple particle system to create something that looks like an explosion. Although it isn't of commercial quality the effect is quite effective. This program shows an easy way to make simple explosions using OpenGL. Press SPACE to blow the rotating cube to pieces! Use the menu to toggle between normalized speed vectors and non-normalized speed vectors. This program uses an extremely simple particle system to create an explosion effect. Each particle is moved from the origin towards a random direction and, if activated, a random speed. The color of the particles are changed from white to orange to red to create a glowing effect. The GL _POINTS primitive is used to render the particles.

The debris is similar to the particles, with the addition of orientation and orientation speed. A point light source is placed in the center of the explosion.

KEYWORDS: *openGL,GL_points, GL_operations, SPACE.*

1. INTRODUCTION

Introduction to OpenGL

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

Here presents a global view of how OpenGL works; it contains the following major sections:

- "OpenGL Fundamentals" briefly explains basic OpenGL concepts, such as what a graphic primitive is and how OpenGL implements a client-server execution model.
- "Basic OpenGL Operation" gives a high-level description of how OpenGL processes data and produces a corresponding image in the frame buffer.

OpenGL Fundamentals

This section explains some of the concepts inherent in OpenGL.

Primitives and Commands

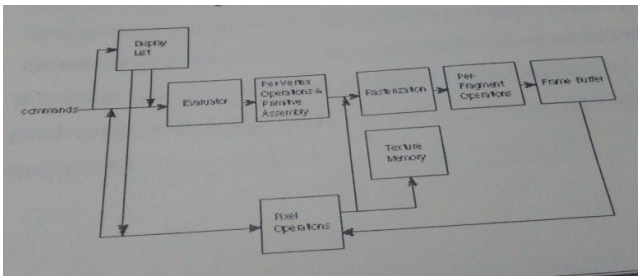
OpenGL draws *primitives-points,* line segments, or polygons-subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are

specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. Primitives are defined by a group of one or more *vertices.* A vertex defines a Pointand endpoint of a line,or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normal, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be *clipped* so that a particular primitive fits within a specified region; in this case, vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents. Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that's consistent with complete execution of all previously issued OpenGL commands. Procedural versus Descriptive OpenGL provides you with fairly direct control over the fundamental operations of two- and three-dimensional graphics. This includes specification of such parameters as transformation matrices, lighting equation coefficients, antialiasing methods, and pixel update operators. However, it doesn't provide you with a means for describing or modeling complex geometric objects. Thus, the OpenGL commands you Issue specify how a certain result should be produced (what procedure should be followed) rather than what exactly that result should look like. That is, OpenGL is fundamentally procedural rather than descriptive. Because of this procedural nature, it helps to know how OpenGL works-the order in which it carries out its operations, for example-in order to fully understand how to use it.

EXECUTION MODEL

The model for interpretation of OpenGL commands is client- server. An application (the client) issues commands, which are interpreted and processed by OpenGL (the server). The server mayor may not operate on the same computer as the client. In this sense, OpenGL is network-transparent. A server can maintain several GL *contexts,* each of which is an encapsulated GL state. A client can connect to anyone of these contexts. The required network protocol can be implemented by augmenting an already existing protocol (such as that of the X Window System) or by using an independent protocol. No OpenGL commands are provided for obtaining user input. The effects of OpenGL commands on the frame buffer are ultimately controlled by the window system that allocates frame buffer resources. The window system determines which portions of the frame buffer OpenGL may access at any given time and communicates to OpenGL how those portions are structured. Therefore, there are no OpenGL commands to configure the frame buffer or initialize OpenGL. Frame buffer configuration is done outside of OpenGL in conjunction with the window system; OpenGL initialization takes place when the window system allocates a window for OpenGL rendering.

Basic OpenGL Operation



The figure shown above gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages. As shown by the first block in the diagram, rather

than having all commands proceeds immediately through the pipeline, you can choose to accumulate some of them in a *display list* for processing at a later time. The *evaluator* stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. During the next stage, *per-vertex operations and primitive assembly,* OpenGL processes geometric primitives- points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.

Rasterization produces a sense of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each *fragment* so produced is fed into the last stage, *per-fragment operations,* which perform the final operations on the data before it's stored as pixels in the *frame buffer.* These operations include conditional updates to the frame buffer based on incoming andpreviously stored z-values (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values. Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the *pixel operations* stage. The result of this stage is either stored as *texture memory,* for use in the Rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data. All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

Explaination of Particle System

Particle systems are collections of particles, typically point masses, In which the dynamic behavior of the particles can be determined by the solution of sets of coupled differential equations. Particle systems have been used to generate a wide variety of behaviors in a number of fields. In fluid dynamics, people use particle systems to model turbulent behavior .We can simulate the behavior of the system by following a group of particles that is subject to a variety of forces and constraints. We can also use particles to model solid objects. For example, a deformable solid can be modeled as a 3-D array of particles that are held together by springs. When the object is subjected to external forces, the panicles move and their positions approximate the shape of the solid object. Computer graphics practitioners have used particles to model such diverse phenomena as fireworks, the flocking behavior of birds, and wave action. In these applications, the dynamics of the particle system gives the positions of the particles, but at each location we can place a graphical object, rather than a point.

Output

This program shows an easy way to make simple explosion using OpenGL. Press SPACE to blow the rotating cube to pieces. Use the menu to toggle between normalized speed vectors and non-normalized speed vectors.

System Specification

Operating system: Windows XP
Software requirements: openGL, Microsoft visual C++ 6.0

Hardware requirements: Pentium pc with 256 of RAM(min) Mouse and keyboard.

2. CONCLUSION

We have presented explosion demo Program shows an easy way to make simple explosions using OpenGL. This demo shows how to use an extremely simple particle system to create something that looks like an explosion. The GL _POINTS primitive is used to render the particles. The debris is similar to the particles, with the addition of orientation and orientation speed.

REFERENCES

Books referred James D Foley and Edward Angel Website: referred www.opengl.com and www.google.com