Tribhuvan University

Institute of Engineering

Pulchowk Campus

Department of Electronics and Computer Engineering

Computer Graphics

A Project Report On

Nightfall

Submitted To:

Dr. Basanta Joshi

Department of Electronics and Communication Engineering

Submitted By:

Jebish Purbey (074 BEX 412)

Manita Dahal
(074BEX414)

Nimesh Rai(074BEX415)

Paru Hang Rai
(074BEX416)

Kathmandu, Nepal

March 5, 2020

ACKNOWLEDGEMENT

It gives us great pleasure to find this opportunity to express our deepest gratitude and sincerest thanks to our respected Dr Basanta Joshi sir, who provided us a great opportunity to do a project on relevant to the subject matter that is Computer Graphics and also for his proper guidance, support and suggestions throughout this semester as well as for this project. His invaluable suggestions and tutelage have enabled us to undertake and complete this project.

We would also like to express our deepest gratitude and appreciation to our project mentor Mr. Suresh Pokhrel Sir, Mr. Anil Verma Sir at Department of Electronics and Communication Engineering, Pulchowk Campus for giving various valuable suggestions, helpful guidance and encouragement in the successful execution of this project work. His guidance and cooperation have led us to the successful completion of our project titled:" Nightfall". We are highly indebted to him for the way with his valuable tips and suggestion that he accorded to us in every respect of our work.

Likewise, we also humbly extend our sense of gratitude to other faculty members and staff of the Department of Electronics and Computer Engineering, Pulchowk Campus, for providing us their valuable help and time in maintaining a congenial working environment.

Last but not the least, we would also like to extend our appreciation to our dear colleagues and friends, who gave us ideas, motivation and supported us throughout this project journey, throughout the tough times we encountered while working on this project.

Yours' Sincerely

Jebish Purbey - 074BEX412

Manita Dahal - 074BEX414

Nimesh Rai   - 074BEX415

Paru Hang Rai-074BEX416

ABSTRACT

In present time, computer graphics is becoming an integral part in various fields such as films, video games, photography, and animation production as well as other specialized digital applications. Though the use of computer graphics, we have been to emulate and display images of an object which look more genuine than the object itself. Various applications have been developed and are in the process of completion, which provides an interface to user for using computer graphics for producing the required image.

OpenGL is an API (Application Programming Interface) that acts as a platform for providing a large group of functions that can be used for manipulation of graphics and images. OpenGL by itself is not an API, but just a specification that is developed and maintained by the Khronos Group

Blender is a free and open-source 3D computer graphics software toolset used for creating animated films, visual effects, art, 3D printed models, motion graphics, interactive 3D applications, and computer games. Blender is a free and open-source 3D computer graphics software application for developing animated movies, visual effects, 3D games, and physics simulations. Blender is written in C, C++, and Python, and allows the user to write custom extensions to Blender in any of those programming languages. Blender's features include 3D modeling, UV unwrapping, texturing, raster graphics editing, fluid and smoke simulation, particle simulation, body simulation, sculpting, animating, match moving, rendering, motion graphics, video editing, and compositing.

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. The most basic edition of Visual Studio, the Community edition, is available free of charge. The slogan for Visual Studio Community edition is "Free, fully-featured IDE for students, open-source and individual developers". The currently supported Visual Studio version is 2019.

TABLE OF CONTENTS

S. n	Title	Page No.
1.	Objectives	07
2.	Introduction	08
2.1	Computer Graphics	08
2.2	OpenGL	09
2.3	Legacy OpenGL	10
2.4	SDL	12
2.5	Blender	14
2,6	Microsoft Visual Studios Community 2019	16
2.7	Meteor Storm	17
3	Literature Review	19
3.1	Computer graphics with OpenGl	19
3.2	A project report on computer graphics object modeling	20
3.3	Legacy OpenGL tutorials and articles-Nehe	20
4	Background	21
4.1	Bressenham's Line Algorithm	21
4.2	Translations	22
4.3	Vectors in C++	23
4.4	3D Viewing Pipeline	23
4.5	Depth Buffer and Depth Test -Z test	24
4.6	Blending	25
4.7	Texturing	26
4.8	Texture Filter	28
4.9	Mipmapping	28

S. n	Title	Page No.
4.10	Anti-Aliasing	29
4.10.1	Multisampling Anti-Aliasing	29
4.11	Lighting in computer Graphics	29
4.11.1	Diffuse	30
4.11.2	Ambient	30
4.11.3	Specular	30
4.12	Illumination MODELS	31
5	Methodology	33
5.1	Tools and Environment Used	33
5.1.1	Minimum Hardware Requirements	33
5.1.1.1	For Blender	33
5.1.1.2	For Visual Studio 2019	33
5.1.2	Minimum Software Requirement	33
5.1.2.1	For Blender	33
5.1.2.2	For Visual Studio 2019	33
5.1.2.3	For SDL	34
5.2	Methods	34
5.2.1	Object Loader	35
5.2.2	SKYBOX	36
5.2.3	Camera Movement	37
5.2.4	Particle Effect	38
6	Result	39
7	Discussion	41
7.1	Advantages	41
7.2	Limitations	42

S.n.	Title	Page No.
8	Conclusion	42
9	References	43
10	Bibliography	44

1. OBJECTIVES

The main objective of our project is to design a scenery for simulation of a meteor flying across the night sky in the background of a mountain using the software discussed above. We also aim to create a house along with a bench and small forest as objects in order to create a realistic environment as well as to make sure that the mountain doesn't look empty and lifeless. Learning about the things not present in our textbooks but related to computer graphics, such as use of blender and other software for creation of 3D objects and environment along with skybox and various other tools which help in improving the quality of any graphics designing activity is also one of our objectives of this project. Here, Blender was primarily used for object creation, texturing etc. while visual studio was used for creation of an interface for use of OpenGL. OpenGL itself was used for Rendering the final scene and for using power of Graphics processing Unit instead of main processor for better performance. .

We aim to fulfil the following basic objectives during and after the fulfilment of the project:

- Learn and understand about 3D creation software and their functioning.
- Design "Nightfall", a fully working 3D model of a meteor travelling across a night sky near a mountain.
- Creation of objects of varying scale and combining them to produce a meaningful environment.
- Understanding about basic legacy lighting and surface rendering through practical applications.
- Learning and implementation of camera movement to see the objects through various angles.
- Become more familiar to OpenGL and its functioning and usage and apply it in current and related future projects.
- Develop research skills and team co-ordination for better project working in future.

2. INTRODUCTION

2.1 Computer Graphics

Computer Graphics involves technology to access. The Process transforms and presents information in a visual form. The role of computer graphics insensible. In today life, computer graphics has now become a common element in user interfaces, T.V. commercial motion pictures.

Computer Graphics is the creation of pictures with the help of a computer. The end product of the computer graphics is a picture it may be a business graph, drawing, and engineering.

Computer graphics is a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Although the term often refers to the study of three-dimensional computer graphics, it also encompasses two-dimensional graphics and image processing. Computer graphics is the discipline of generating images with the aid of computers. Today, computer graphics is a core technology in digital photography, film, video games, cell phone and computer display, and many specialized applications. A great deal of specialized hardware and software has been developed, with the displays of most devices being driven by computer graphics hardware. It is a vast and recently developed area of computer science. The phrase was coined in 1960 by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, or typically in the context of film as CGI.

Some topics in computer graphics include user interface design, sprite graphics, rendering, ray tracing, geometry processing, computer animation, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization, image processing, computational photography, scientific visualization, computational geometry and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, physics, and perception.

Computer graphics is responsible for displaying art and image data effectively and meaningfully to the consumer. It is also used for processing image data received from the physical world. Computer graphics development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design in general.

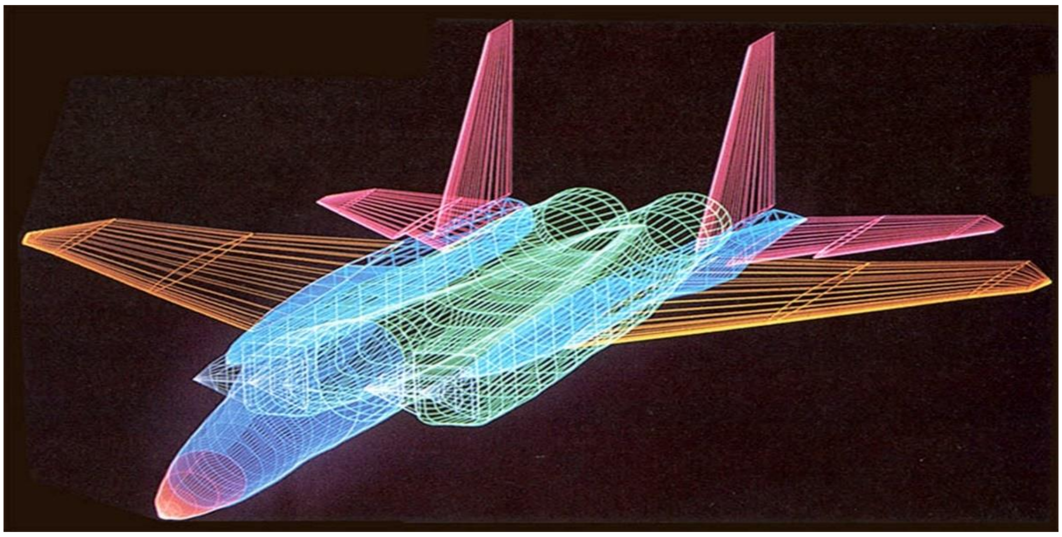


Figure 1 Use of Computer Graphics for Designing

2.2 OpenGL

Open Graphics Library (OpenGL) is a cross-language, cross-OS, cross-platform application programming interface (API) that contains a large set of functions for creating and manipulating both 3D and 2D graphics images. It is also known as graphics rendering API which generates high-quality color images composed of geometric and image primitives. This interface consists of 250 distinct commands (200 in core OpenGL and 50 in OpenGL Utility Library) to produce interactive 3D applications.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware.

The API is defined as a set of functions which may be called by the client program, alongside a set of named integer constants (for example, the constant GL_TEXTURE_2D, which corresponds to the decimal number 3553). Although the function definitions are superficially similar to those of the programming language C, they are language-independent. As such, OpenGL has many language bindings, some of the most noteworthy being the JavaScript binding WebGL (API, based on OpenGL ES 2.0, for 3D rendering from within a web browser); the C bindings WGL, GLX and CGL; the C binding provided by iOS; and the Java and C bindings provided by Android.

In addition to being language-independent, OpenGL is also cross-platform. The specification says nothing on the subject of obtaining, and managing an OpenGL context, leaving this as a detail of the underlying windowing system. For the same reason, OpenGL is purely concerned with rendering, providing no APIs related to input, audio, or windowing.

OpenGL is an evolving API. New versions of the OpenGL specifications are regularly released by the Khronos Group, each of which extends the API to support various new features. The details of each version are decided by consensus between the Group's members, including graphics card manufacturers, operating system designers, and general technology companies such as Mozilla and Google.

Figure 2OpenGL Logo

2.3 Legacy OpenGL

As the name itself suggests, Legacy OpenGL is an older and dated version of the modern OpenGL that we are currently familiar with. In 2008, version 3.0 of the OpenGL specification was released. With this revision, the Fixed Function Pipeline as well as most of the related OpenGL functions and constants were declared deprecated. These deprecated elements and concepts are now commonly referred to as **legacy OpenGL**. Legacy OpenGL is still supported by certain implementations that support core OpenGL 3.1 or higher and the GL ARB compatibility extension. Implementations that do not expose this extension do only offer features defined in the core OpenGL specification the implementation is based upon.

Legacy OpenGL is a more general-purpose API that provides the users with functions like “vertex” “begin primitive”, matrix stack, and so on. It also relies on existence of predefined lighting systems (fixed function pipeline). Significant part of the legacy OpenGL is still be available under “compatibility” profile, while the more powerful new API is placed under “core” profile. It means that the users don’t have to restrict themselves to new functions only. Both AMD and NVIDIA provide backwards-compatible implementations at least on Windows and Linux. Apple does only provide an implementation of the core profile and supports core OpenGL 3.2 on Mac OSX. Intel provides an implementation for Windows up to OpenGL 3.1 with Sandy Bridge CPUs and OpenGL 4.0 with Ivy Bridge CPUs.

The areas that are affected by the removal of functionality from core OpenGL 3.1 and higher are as follows:

- Fixed-function vertex and fragment processing. These stages of the rendering pipeline are now the developer's responsibility and can be fully programmed using shaders.
- Immediate-mode vertex attribute specification, client-side vertex arrays. Vertex Attributes must be stored in one or more Buffer Objects, or be specified explicitly using glVertexAttrib*(). Since OpenGL 3.2 using Vertex Array Objects is mandatory.
- The GL_QUAD and GL_POLYGON primitive types.
- The matrix stack and related matrix manipulation functions. This includes projection, model-view and texture matrices. Corresponding built-in GLSL functions have also been removed. Managing matrices and sending data to shaders is now the developer's responsibility.
- Fixed-function lighting, materials and color materials, fixed-function shadow mapping and bump mapping. Everything possible with functions corresponding to these areas can easily be emulated with shaders.
- Accumulation and auxiliary buffers. If additional buffers are needed Framebuffer Objects can be used.
- Versions 1.10 and 1.20 of the GLSL.

2.4 SDL

Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D. It is used by video playback software, emulators, and popular games including Valve's award winning catalog and many Humble Bundle games.

SDL officially supports Windows, Mac OS X, Linux, iOS, and Android. Support for other platforms may be found in the source code.

SDL is written in C, works natively with C++, and there are bindings available for several other languages, including C# and Python.

The Simple DirectMedia Layer library (SDL) is a general API that provides low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D framebuffer across multiple platforms.

Video

- 3D graphics:
    - SDL can be used in combination with the OpenGL API or Direct3D API for 3D graphics
- Accelerated 2D render API:
    - Supports easy rotation, scaling and alpha blending, all accelerated using modern 3D APIs
    - Acceleration is supported using OpenGL and Direct3D, and there is a software fallback
- Create and manage multiple windows

Input Events

- Events and API functions provided for:
    - Application and window state changes
    - Mouse input
    - Keyboard input

    - Joystick and game controller input
    - Multitouch gestures
- Each event can be enabled or disabled with SDL_EventState()
- Events are passed through a user-specified filter function before being posted to the internal event queue
- Thread-safe event queue

Audio

- Set audio playback of 8-bit and 16-bit audio, mono stereo or 5.1 surround sound, with optional conversion if the format is not supported by the hardware
- Audio runs independently in a separate thread, filled via a user callback mechanism
- Designed for custom software audio mixers, but SDL_mixer provides a complete audio/music output library

File I/O Abstraction

- General purpose abstraction for opening, reading and writing data
- Built-in support for files and memory

Shared Object Support

- Load shared objects (DLL on Windows,. dylib on Mac OS X, .so on Linux)
- Lookup functions in shared objects

Threads

- Simple thread creation API
- Simple thread local storage API
- Mutexes, semaphores and condition variables
- Atomic operations for lockless programming

Timers

- Get the number of milliseconds elapsed
- Wait a specified number of milliseconds
- Create timers that run alongside your code in a separate thread
- Use high resolution counter for profiling

2.5 Blender

Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. Advanced users employ Blender's API for Python scripting to customize the application and write specialized tools; often these are included in Blender's future releases. Blender is well suited to individuals and small studios who benefit from its unified pipeline and responsive development process.

Blender is cross-platform and runs equally well on Linux, Windows, and Macintosh computers. Its interface uses OpenGL to provide a consistent experience. To confirm specific compatibility, the list of supported platforms indicates those regularly tested by the development team.

As a community-driven project under the GNU General Public License (GPL), the public is empowered to make small and large changes to the code base, which leads to new features, responsive bug fixes, and better usability. Blender is free for everyone: Blender is everyone's own 3D software.

Some of the features of Blender are as follows:

- Support for a variety of geometric primitives, including polygon meshes, fast subdivision surface modeling, Bezier curves, NURBS surfaces, metaballs, multi-res digital sculpting (including dynamic topology, maps baking, re-meshing, re-symmetrize, decimation), outline font, and a new n-gon modeling system called B-mesh.
- Internal render engine with scan line rendering, indirect lighting, and ambient occlusion that can export in a wide variety of formats.

- Integration with a number of external render engines through plugins.
- Key framed animation tools including inverse kinematics, armature (skeletal), hook, curve and lattice-based deformations, shape animations, non-linear animation, constraints, and vertex weighting.
- Simulation tools for soft body dynamics including mesh collision detection, LBM fluid dynamics, smoke simulation, Bullet rigid body dynamics, and an ocean generator with waves.
- A particle system that includes support for particle-based hair.
- Modifiers to apply non-destructive effects.
- Python scripting for tool creation and prototyping, game logic, importing/exporting from other formats, task automation and custom tools.
- Basic non-linear video/audio editing.
- A fully integrated node-based compositor within the rendering pipeline accelerated with OpenGL.
- Procedural and node-based textures, as well as texture painting, projective painting, vertex painting, weight painting and dynamic painting.
- Real-time control during physics simulation and rendering.
- Camera and object tracking.
- Grease Pencil tools for 2D animation within a full 3D pipeline.

2.6 Microsoft Visual Studio Community 2019

Visual Studio, also known as Microsoft Visual Studio and VS, is an integrated development environment for Microsoft Windows. It is a tool for writing computer programs, websites, web apps, and web services. It includes a code editor, debugger, GUI design tool, and database schema designer, and supports most major revision control systems. It is available in both a free "Community" edition as well as a paid commercial version. Some of the programming languages supported by Visual studio are C, C++, C#, F#, Python, HTML/XHTML, and JavaScript etc.

The Community edition was announced on 12 November 2014, as a new free version, with similar functionality to Visual Studio Professional. Visual Studio Community supports multiple languages, and provides support for extensions. Individual developers have no restrictions on their use of the Community edition. The following uses also allow unlimited usage: contributing to Open Source projects, academic research, in a classroom learning environment and for developing and testing device drivers for the Windows operating system. Visual Studio Community is oriented towards individual developers and small teams.

Visual Studio does not support any programming language, solution or tool intrinsically; instead, it allows the plugging of functionality coded as a VSPackage. When installed, the functionality is available as a *Service*. The IDE provides three services: SVsSolution, which provides the ability to enumerate projects and solutions; SVsUIShell, which provides windowing and UI functionality (including tabs, toolbars, and tool windows); and SVsShell, which deals with registration of VSPackages. In addition, the IDE is also responsible for coordinating and enabling communication between services. All editors, designers, project types and other tools are implemented as VSPackages. Visual Studio uses COM to access the VSPackages. The Visual Studio SDK also includes the *Managed Package Framework* (*MPF*), which is a set of managed wrappers around the COM-interfaces that allow the Packages to be written in any CLI compliant language. However, MPF does not provide all the functionality exposed by the Visual Studio COM interfaces. The services can then be consumed for creation of other packages, which add functionality to the Visual Studio IDE.

Some of the features of Visual Studio are as follows:

- Visual Studio includes a code editor that supports syntax highlighting and code
  completion using IntelliSense for variables, functions, methods, loops,
  and LINQ queries.
- Visual Studio includes a debugger that works both as a source-level debugger and as a machine-level debugger.
- Visual Studio includes a host of visual designers to aid in the development of applications. These tools include Windows Form Designer, WPF Designer, Web designer etc.
- Visual Studio allows developers to write extensions for Visual Studio to extend its capabilities. These extensions "plug into" Visual Studio and extend its functionality.
- Visual Studio includes tools like Properties Editor, Object Browser, Solution Explorer, Data Explorer, Server Explorer and Visual Studio Tools for Office etc.

2.7 METEOR STORM

A meteor shower is a celestial event in which a number of meteors are observed to radiate, or originate, from one point in the night sky. These meteors are caused by streams of cosmic debris called meteoroids entering Earth's atmosphere at extremely high speeds on parallel trajectories.

Very intense or unusual meteor showers are known as meteor outbursts and meteor storms, which produce at least 1,000 meteors an hour, most notably from the Leonids. The Leonids (/ˈliːənɪdz/ LEE-ə-nidz) are a prolific meteor shower associated with the comet Tempel–Tuttle, which are also known for their spectacular meteor storms that occur about every 33 years.

Night Falls is our proposed animated video that will showcase a meteor storm in the night sky. This is a proposed project which will show the night sky full of meteors as animated visual. The meteors will be fully simulated and the impact of meteor strikes will also be shown. Meteor storms are a spectacular phenomenon of nature that most often occurs because of debris from a big passing comet.

Night Falls will be based on the same concept where a big passing by comet will cause a meteor storm in Earth’s atmosphere. The aim is to recreate a very photo realistic meteor shower with some beauty auras for an imaginary Comet.

The name “Night Falls” is chosen so as to signify the meteor outburst phase as many heavenly bodies seem to fall during this moment giving a pseudo illusion that the night sky is about to collapse.

Figure the Geminids Meteor Storm

3. LITERATURE REVIEW

The project is titled Nightfalls as meteors are seen always at night because intense sun light at the day prevents any sightings as of such. As for the project inspirations, few of the articles were reviewed. The project takes some of the outcomes from the following Literature articles and papers:

- **A study on the use of OpenGL in window systems** -Johan Persson, Department of Software Engineering and Computer Science Blekinge Institute of Technology
- **Advanced Computer Graphics using OpenGL - Sven Maerivoet :** In this manual, a general framework for easily creating OpenGL-applications is considered. This framework is written in C++ and relies heavily on the concept of Object-Oriented Programming (OOP). General knowledge of encapsulation, inheritance and polymorphism is assumed.Various aspects are discussed, ranging from a general introduction and detailed explanations of class-methods to an in-depth treatment of the ray tracer, which was (partially) built using this framework.Much of the work is based on the book [Hil01], the OpenGL-standard as defined in [SA99]. The GLUT-API as defined in [Kil96] and the 'standard-bible' [FvDFH90]. The programming-techniques used, are based on those as presented in [Str97] and [CL95].

3.1 Computer graphics with OpenGL

-Donald Hearn, M. Pauline Baker, Warren Carithers

Reflecting the rapid expansion of the use of computer graphics and of C++ as a programming language of choice for implementation, this book converts all programming code into the C++ language. This new edition is a complete revision, bringing the text up to date with current advances in computer graphics technology and applications. Assuming readers have no prior familiarity with computer graphics, the authors—both authorities in their field—present basic principles for design, use, and understanding of computer graphics systems using their well-known, and accessible writing style. It includes an exploration of GLUT and other graphics libraries and covers topics such as distributed ray tracing, radiosity, physically based modeling, particle systems, and visualization techniques. For professionals in any area of computer graphics: CAD, Animation, Software Design, etc.

3.2 A PROJECT REPORT ON COMPUTER GRAPHICS OBJECT MODELLING

Manish Jayswal 071BCT520
Pranil GC 071BCT524
Shailesh Kandel 071BCT537
A sample report from our seniors where many different aspects of computer graphics using c++ has been described and also many other findings which have been useful to us. From their introduction: This project is thus an application of many of the graphics algorithms and processes we
studied in the syllabus. This project contains a table tennis board with bats and a ball. All the objects are created in 3D plane using object-oriented approach.

3.3 Legacy OpengL tutorials and Articles – Nehe

This tutorial articles have everything about computer graphics possible with legacy opengl in any computer. The articles discuss about the use of legacy opengl instead of modern for beginners and starts from the basic of initializing the opengl. The Articles contains step by step guidance about different prospects of designing computer graphics with Opengl including creation of windows, setting of color buffers, using matrices, texturing images, loading objects, particles system, volumetric fogs and many more.
Most of the functions implemented in this project has been influenced by these tutorials. The tutorials all uses Glaux, an outdate opengl toolkit while this project has been done using SDL library.

4. BACKGROUND THEORY

4.1 Bresenham's line algorithm

Bresenham's line algorithm is a line drawing algorithm that determines the points of an *n*-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. It is commonly used to draw line primitives in a bitmap image (e.g. on a computer screen), as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. It is an incremental error algorithm. It is one of the earliest algorithms developed in the field of computer graphics. An extension to the original algorithm may be used for drawing circles.

While algorithms such as Wu's algorithm are also frequently used in modern computer graphics because they can support antialiasing, the speed and simplicity of Bresenham's line algorithm means that it is still important. The algorithm is used in hardware such as plotters and in the graphics chips of modern graphics cards. It can also be found in many software graphics libraries. Because the algorithm is very simple, it is often implemented in either the firmware or the graphics hardware of modern graphics cards. The label "Bresenham" is used today for a family of algorithms extending or modifying Bresenham's original algorithm.

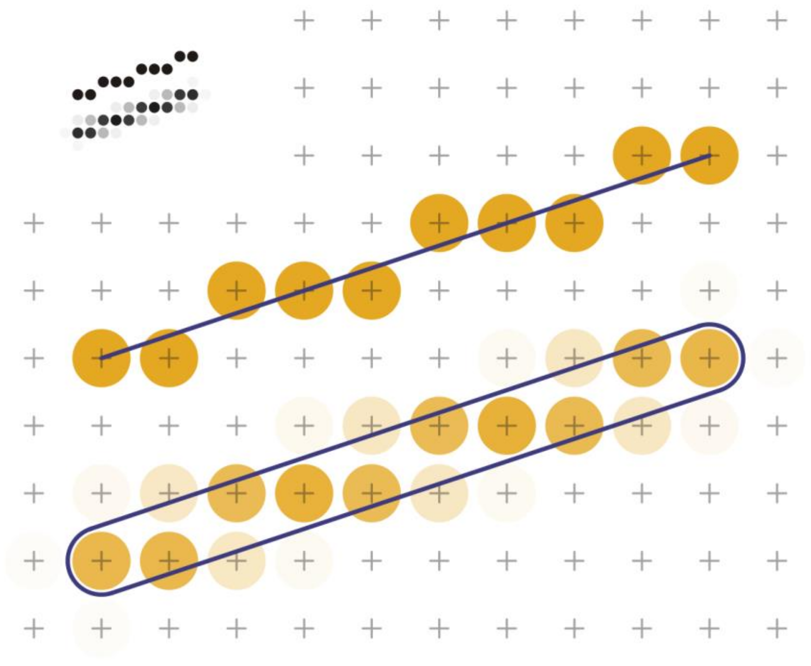


Figure 3Bresenham's approach of line drawing

Opengl uses Bresenham's algorithm for line drawing though all the calculations are done by itself. Line drawing approach using opengl:

glBegin (GL_LINES);

glVertices3f (x1, y1, z1);

glVertices3f (x1, y1, z1);

glEnd ();

Same line drawing algorithm are used for drawing polygons using more vertices.

4.2 TRANSFORMATIONS

Transformations represent changing the shape, size, position or orientations or all of them of an image. Transformations are fairly simple even for human calculations. Three basic transformations are:

Translations: Changing the position coordinates of object is translation.

Rotation: Rotating an object about some axes by certain angle is rotation.

Scaling: Changing the size of the object by any arbitrary factor is scaling.

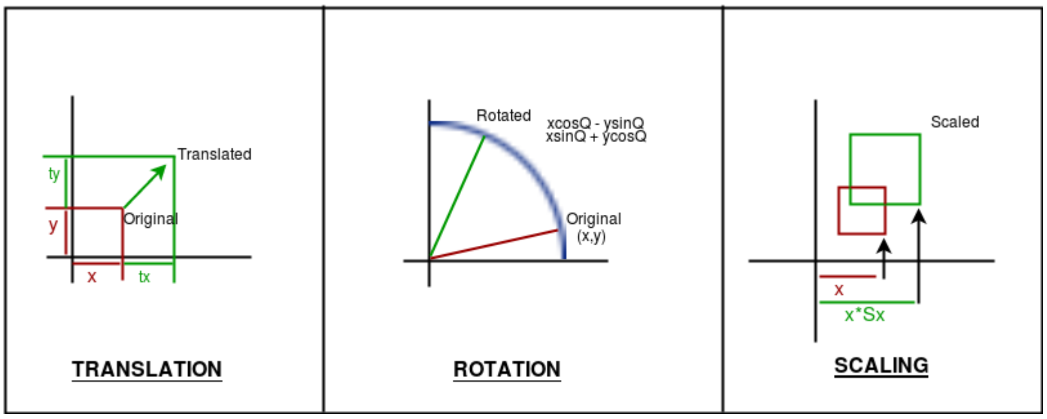


Figure 4 Basic Transformations

Opengl can do basic transformations just with one function calls. They are:

Translation : glTranslatef(tx,ty,tz); //for 3D transformations

Rotation: glRotatef(angle,axes-x,axes-y,axes-z);

Scaling: glScalef(Sx,Sy,Sz);

4.3 Vectors in C++

Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes there may be a need of extending the array. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time.

Vector is better for frequent insertion and deletion whereas Arrays are much better suited for frequent access of elements scenario. Vector occupies much more memory in exchange for the ability to manage storage and grow dynamically whereas Arrays are memory efficient data structure.

vector::push_back : Add element at the end
Adds a new element at the end of the vector, after its current last element. The content of *val* is copied (or moved) to the new element. This effectively increases the container size by one, which causes an automatic reallocation of the allocated storage space if -and only if- the new vector size surpasses the current vector capacity.

4.4 3D Viewing Pipeline:

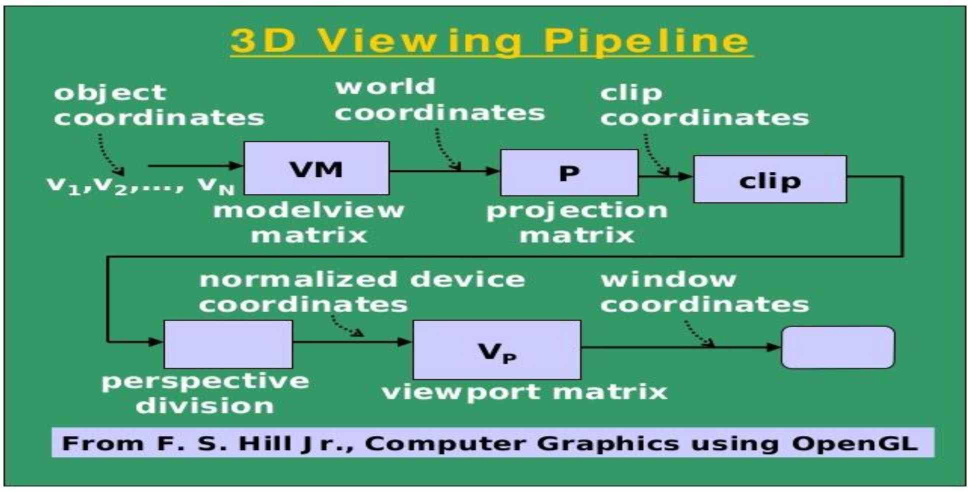
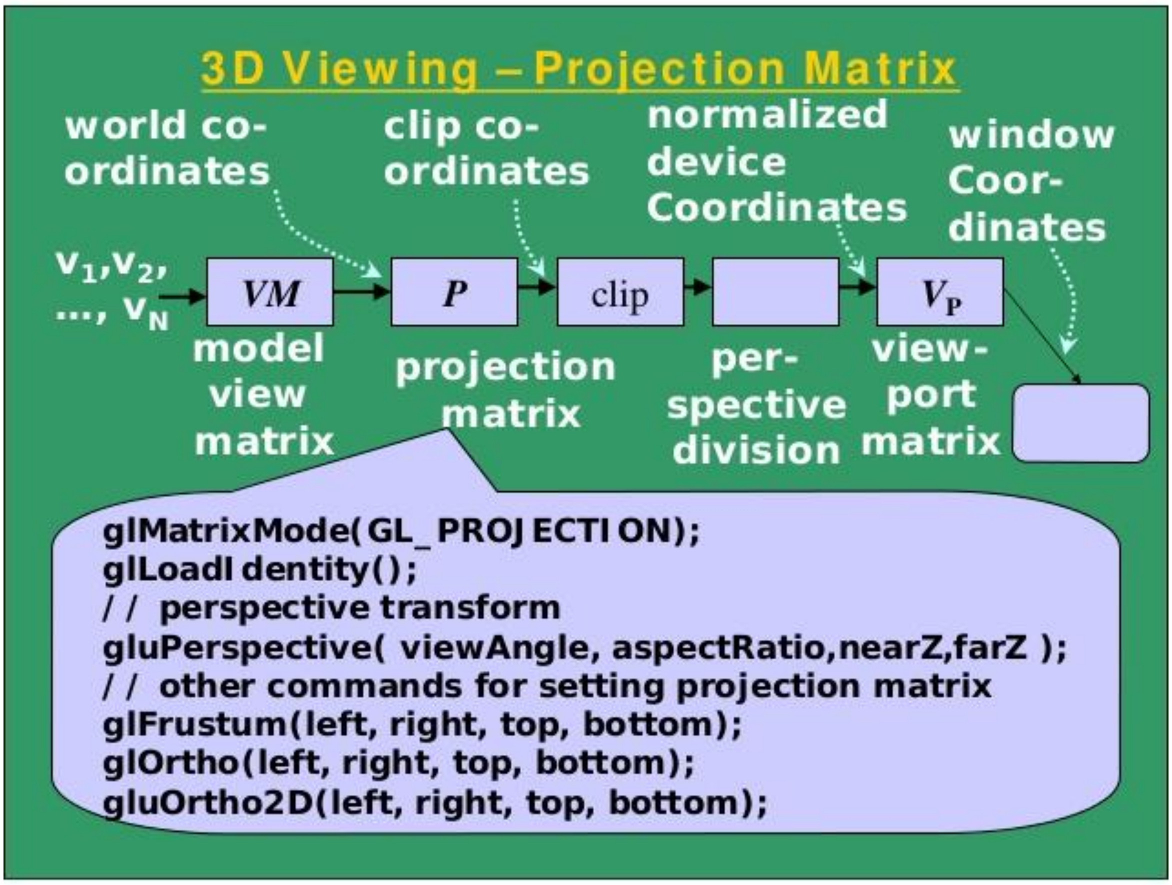


Figure 5Viewing pipeline OPENGL

A series of operations are applied to the OpenGL matrices, in order to create a 2D representation from 3D geometry. Processes can be broken up into four main areas:

1. The movement of the object is called a modeling transformation.
2. The movement of camera is called a viewing transformation.
3. The conversion from 3D to 2D is called a projection transformation.
4. The 2D picture plane. Which is mapped to the screen viewport, is called viewport transformation.



```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
// perspective transform
gluPerspective( viewAngle, aspectRatio,nearZ,farZ );
// other commands for setting projection matrix
glFrustum(left, right, top, bottom);
glOrtho(left, right, top, bottom);
gluOrtho2D(left, right, top, bottom);
```

Figure 6:Opengl 3D viewing pipeline

4.5 Depth Buffer and Depth Test – Z test

The depth-buffer is a buffer that, just like the color buffer (that stores all the fragment colors: the visual output), stores information per fragment and has the same width and height as the color buffer. The depth buffer is automatically created by the windowing system and stores its depth values as 16, 24- or 32-bit floats. In most systems, depth buffer will have a precision of 24 bits.

When depth testing is enabled, OpenGL tests the depth value of a fragment against the content of the depth buffer. OpenGL performs a depth test and if this test passes, the fragment is rendered and the depth buffer is updated with the new depth value. If the depth test fails, the fragment is discarded.

Z-buffering is a way of keeping track of the depth of every pixel on the screen. The depth is an increasing function of the distance between the screen plane and a fragment that has been drawn. That means that the fragments on the sides of the cube further away from the viewer have a higher depth value, whereas fragments closer have a lower depth value.

If this depth is stored along with the color when a fragment is written, fragments drawn later can compare their depth to the existing depth to determine if the new fragment is closer to the viewer than the old fragment. If that is the case, it should be drawn over and otherwise it can simply be discarded. This is known as depth testing.

Depth testing for opengl: glEnable (GL_DEPTH_TEST);

4.6 Blending:

Blending is OpenGL's mechanism for combining color already in the framebuffer with the color of the incoming primitive. The result of this combination is then stored back in the framebuffer. Blending is frequently used to simulate translucent physical materials. One example is rendering the smoked glass windshield of a car. The driver and interior are still visible, but they are obscured by the dark color of the smoked glass.

OpenGL doesn't support a direct interface for rendering translucent (partially opaque) primitives. However, you can create a transparency effect with the blend feature and carefully ordering your primitive data. You might also consider using screen door transparency.

An OpenGL application typically enables blending as follows:

glEnable (GL_BLEND); glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

After blending is enabled, as shown above, the incoming primitive color is blended with the color already stored in the framebuffer. glBlendFunc () controls how this blending occurs. The typical use described above modifies the incoming color by its associated alpha value and modifies the destination color by one minus the incoming alpha value. The sum of these two colors is then written back into the framebuffer.

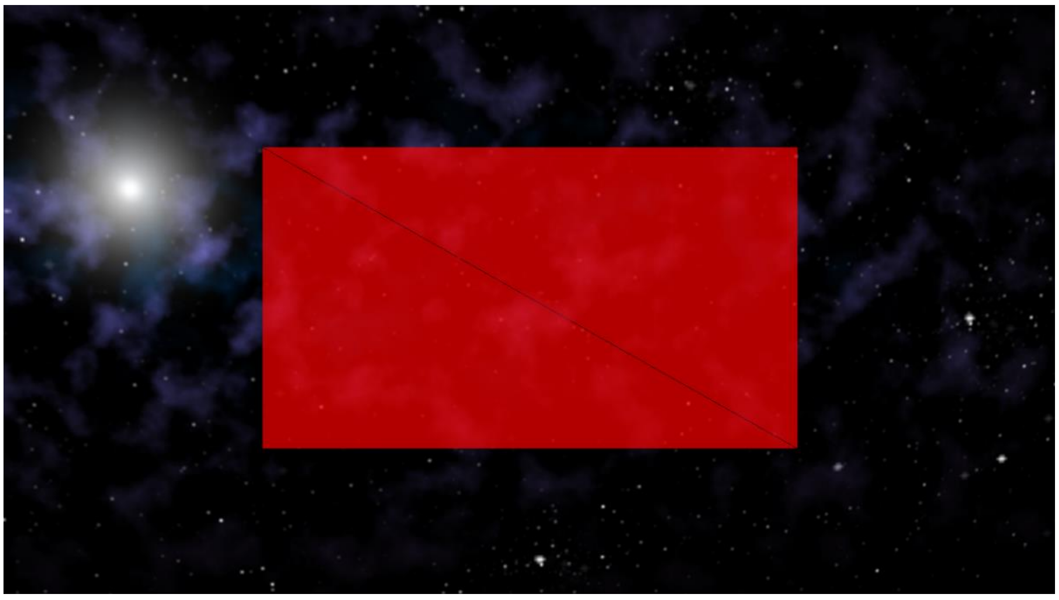


Figure 7Blending of a quad

4.7 Texturing:

Texture mapping is a method for defining high frequency detail, surface texture, or color information on a computer-generated graphic or 3D model. The original technique was pioneered by Edwin Catmull in 1974.

Texture mapping originally referred to diffuse mapping, a method that simply mapped pixels from a texture to a 3D surface ("wrapping" the image around the object). In recent decades, the advent of multi-pass rendering, multitexturing, mipmaps, and more complex mappings such as height mapping, bump mapping, normal mapping, displacement mapping, reflection mapping, specular mapping, occlusion mapping, and many other variations on the technique (controlled by a materials system) have made it possible to simulate near-photorealism in real time by vastly reducing the number of polygons and lighting calculations needed to construct a realistic and functional 3D scene.

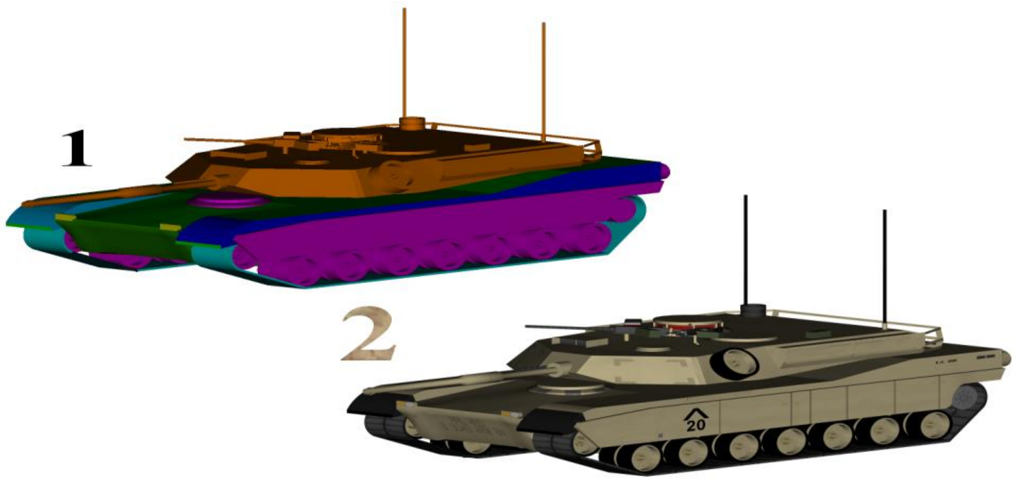


Figure 8 3D models with and without texturing

Texturing in opengl: Texture Mapping in openGL is fairly easy. Only the texture binding and normalized vertices of the polygon surface is required:

Example from our project:

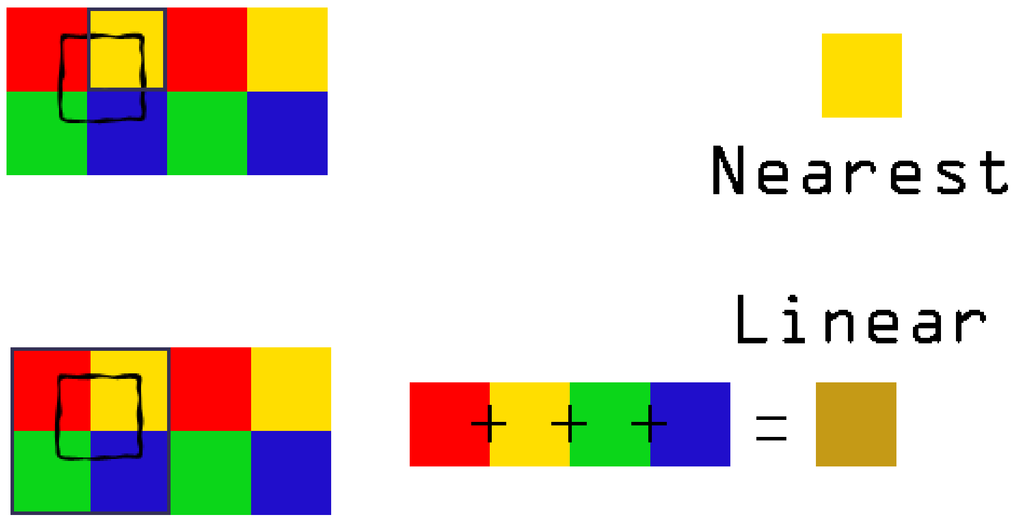
```
skybox [SKY_BOTTOM] = loadTexture("down.png");

glBindTexture (GL_TEXTURE_2D, skybox [SKY_BOTTOM]);
   glBegin (GL_QUADS); //bottom face
   glTexCoord2f (1, 1);
   glVertex3f (size / 2, -size / 2, size / 2);
   glTexCoord2f (0, 1);
   glVertex3f (-size / 2, -size / 2, size / 2);
   glTexCoord2f (0, 0);
   glVertex3f (-size / 2, -size / 2, -size / 2);
   glTexCoord2f (1, 0);
   glVertex3f (size / 2, -size / 2, -size / 2);
   glEnd ();
```

4.8 Textures filter:

Filtering is the process of accessing a particular sample from a texture. There are two cases for filtering: minification and magnification. Magnification means that the area of the fragment in texture space is smaller than a texel, and minification means that the area of the fragment in texture space is larger than a texel. Filtering for these two cases can be set independently.

The magnification filter is controlled by the GL_TEXTURE_MAG_FILTER texture parameter. This value can be GL_LINEAR or GL_NEAREST. If GL_NEAREST is used, then the implementation will select the texel nearest the texture coordinate; this is commonly called "point sampling". If GL_LINEAR is used, the implementation will perform a weighted linear blend between the nearest adjacent samples.

The minification filter is controlled by the GL_TEXTURE_MIN_FILTER texture parameter.



4.9 Mipmapping:

In computer graphics, mipmaps (also MIP maps) or pyramids are pre-calculated, optimized sequences of images, each of which is a progressively lower resolution representation of the same image. The height and width of each image, or level, in the mipmap is a power of two smaller than the previous level. Mipmaps do not have to be square. They are intended to increase rendering speed and reduce aliasing artifacts. A high-resolution mipmap image is used for high-density samples, such as for objects close to the camera. Lower-resolution images are used as the object appears farther away. This is a more efficient way of downfiltering (minifying) a texture than sampling all texels in the original texture that would contribute to a screen pixel; it is faster to take a constant number of samples from the appropriately downfiltered textures. Mipmaps are widely used in 3D computer games, flight simulators, other 3D imaging systems for texture filtering and 2D as well as 3D GIS software. Their use is known as mipmapping.

4.10 Anti-Aliasing :

Antialiasing is a technique used in computer graphics to remove the aliasing effect. The aliasing effect is the appearance of jagged edges or “jaggies” in a rasterized image (an image rendered using pixels). The problem of jagged edges technically occurs due to distortion of the image when scan conversion is done with sampling at a low frequency, which is also known as Undersampling. Aliasing occurs when real-world objects which comprise of smooth, continuous curves are rasterized using pixels.

Cause of anti-aliasing is Undersampling. Undersampling results in loss of information of the picture. Undersampling occurs when sampling is done at a frequency lower than Nyquist sampling frequency. To avoid this loss, we need to have our sampling frequency at least twice that of highest frequency occurring in the object.

This minimum required frequency is referred to as Nyquist sampling frequency **(fs)**.

4.10.1 Multi Sampling Anti Aliasing :

Legacy opengl doesn’t allow multi sample Anti-Aliasing, but luckily SDL does. So, SDL itself creates a lot of samples for a pixel and then uses their average to create a less jagged image. Example from our project:

```
SDL_GL_SetAttribute (SDL_GL_MULTISAMPLEBUFFERS,1);
//Multisampling Anti Aliasing
SDL_GL_SetAttribute (SDL_GL_MULTISAMPLESAMPLES,16);
```

4.11 Lighting in computer Graphics:

Computer graphics lighting is the collection of techniques used to simulate light in computer graphics scenes. While lighting techniques offer flexibility in the level of detail and functionality available, they also operate at different levels of computational demand and complexity. Graphics artists can choose from a variety of light sources, models, shading techniques, and effects to suit the needs of each application.

4.11.1 Diffuse

Diffuse lighting (or diffuse reflection) is the direct illumination of an object by an even amount of light interacting with a light-scattering surface. After light strikes an object, it is reflected as a function of the surface properties of the object as well as the angle of incoming light. This interaction is the primary contributor to the object's brightness and forms the basis for its color[6].

4.11.2 Ambient

As ambient light is directionless, it interacts uniformly across all surfaces, with its intensity determined by the strength of the ambient light sources and the properties of objects' surface materials, namely their ambient reflection coefficients.

4.11.3 Specular

The specular lighting component gives objects shine and highlights. This is distinct from mirror effects because other objects in the environment are not visible in these reflections. Instead, specular lighting creates bright spots on objects based on the intensity of the specular lighting component and the specular reflection coefficient of the surface.

Only ambient and diffuse lights can be used either together or separately. Though, it is advised to use at least one ambient light source before using diffuse light as unlighted side of the polygon can be really dark if not used.

Light in openGL:

Example from project:

```
glEnable (GL_LIGHTING);
glEnable (GL_LIGHT0);
float pos [] = {10.0,1.0, -2.0,1.0};
glLightfv (GL_LIGHT0, GL_POSITION, pos);
float col2[] = {0.5,0.5,0.5,0.1};
glLightfv (GL_LIGHT1, GL_AMBIENT, col2);
```

4.12 Illumination MODELS:

Illumination models are used to generate the color of an object’s surface at a given point on that surface.

The factors that govern the illumination model determine the visual representation of that surface.

Due to the relationship defined in the model between the surface of the objects and the lights affecting it, illumination models are also called shading models or lighting models.

Legacy opengl only supports flat and gouraud shading. It doesn’t support Phong Shading model.

The difference when it comes to shading is, that to save time in shading legacy openGL only calculates shading values at every vertex position, blending color for every triangle between the 3-color calculated for the vertexes. This method is also called Gouraud shading.

Phong shading on the other hand will calculate lighting values for every pixel rendered on the screen - not just for the vertexes and somehow "guess" all the other lighting values.

Most probably Gouraud shading was chosen as it involves a lot less computation it achieves a relatively realistic looking approximation.

Phong shading may achieve an even more realistic looking approximation but the difference in computation can be very big.

What's missing in legacy opengl to perform fast Phong shading are not any missing or otherwise discarded normals but a way to perform the mentioned per-pixel operations pretty fast.

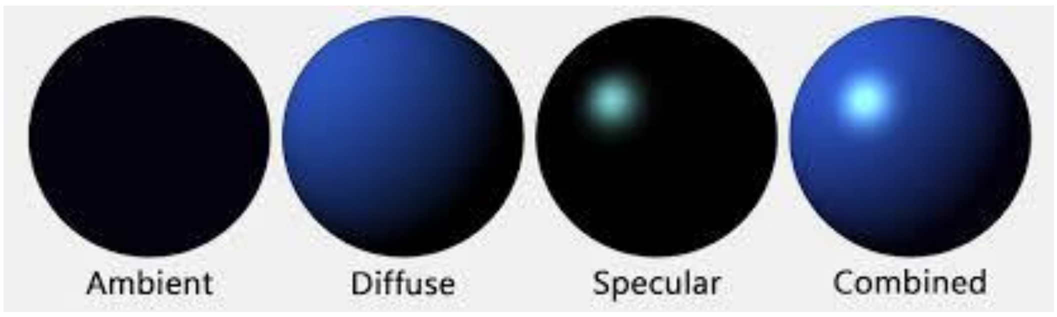


Figure 9 Different illumination conditions

5. METHODOLOGY

5.1 TOOLS AND ENVIRONMENT USED

5.1.1 Minimum Hardware Requirements:

5.1.1.1 For Blender

- 64-bit dual core 2Ghz CPU with SSE2 support
- 4 GB RAM
- Mouse, track pad or pen + tablet
- Graphics card with 1 GB RAM, OpenGL 3.3

5.1.1.2 For Visual Studio 2019

- 1.8Ghz or faster dual core processor
- 2 GB RAM
- 800 MB to 210GB of available hard space
- Video card that supports minimum display of 1280*720

5.1.2 Minimum Software Requirement:

5.1.2.1 For Blender

Blender is cross-platform, it runs on every major operating system:

- Windows 10, 8 and 7
- MacOS 10.12+
- Linux

5.1.2.2 For Visual Studio 2019

- Windows 10
- Windows 8.1
- Windows 7 SP1

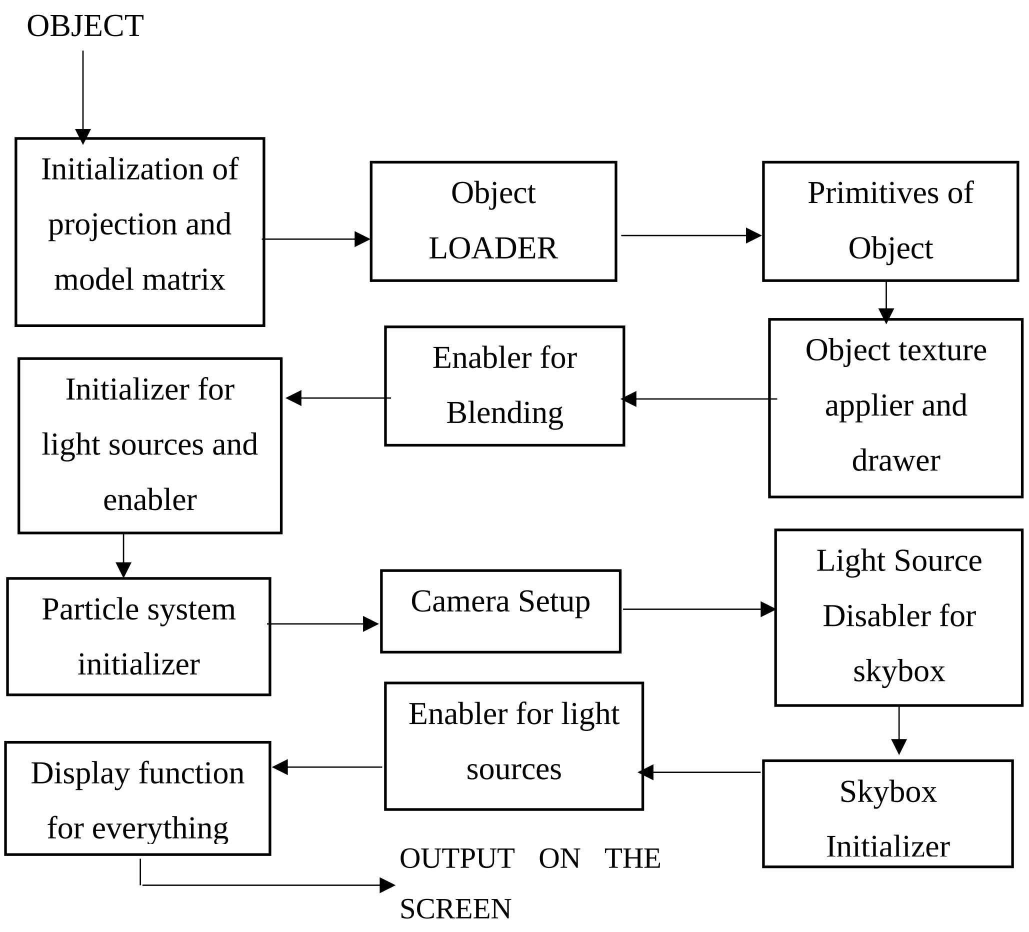
5.1.1.3 For SDL

- Uses Win32 APIs for display, taking advantage of Direct3D for hardware acceleration
- Uses DirectSound and XAudio2 for sound

5.2 Methods:

The project was completed in sequence. First of all, enough research was done for checking out what tools to use and what algorithms to flow. Also, feasibility was checked that whether it was beneficial to use a third-party software for Object drawing or whether to draw the objects manually with opengl primitives drawing techniques.

After all things were considered, legacy opengl, Blender and Visual Studio Community 2019 was chosen for the project completion. For some texture image editing, GIMP- a free open source professional image manipulation tool was used. The basic pipeline of the project methodology can be described as follows:


OBJECT
Initialization of projection and model matrix
Object LOADER
Primitives of Object
Object texture applier and drawer
Enabler for Blending
Initializer for light sources and enabler
Particle system initializer
Camera Setup
Light Source Disabler for skybox
Skybox Initializer
Enabler for light sources
Display function for everything
OUTPUT ON THE SCREEN

Work Division and Block Diagram for divided works:

1. **Jebish Purbey**: House Modelling, Object Loader and source file
2. **Manita Dahal**: Terrain modeling, Bench modelling and Camera movement coding
3. **Nimesh Rai:** Trees Modelling and Skybox Coding
4. **Paru Hang Rai**: Meteor Modelling and Particle System

5.2.1 Object Loader:

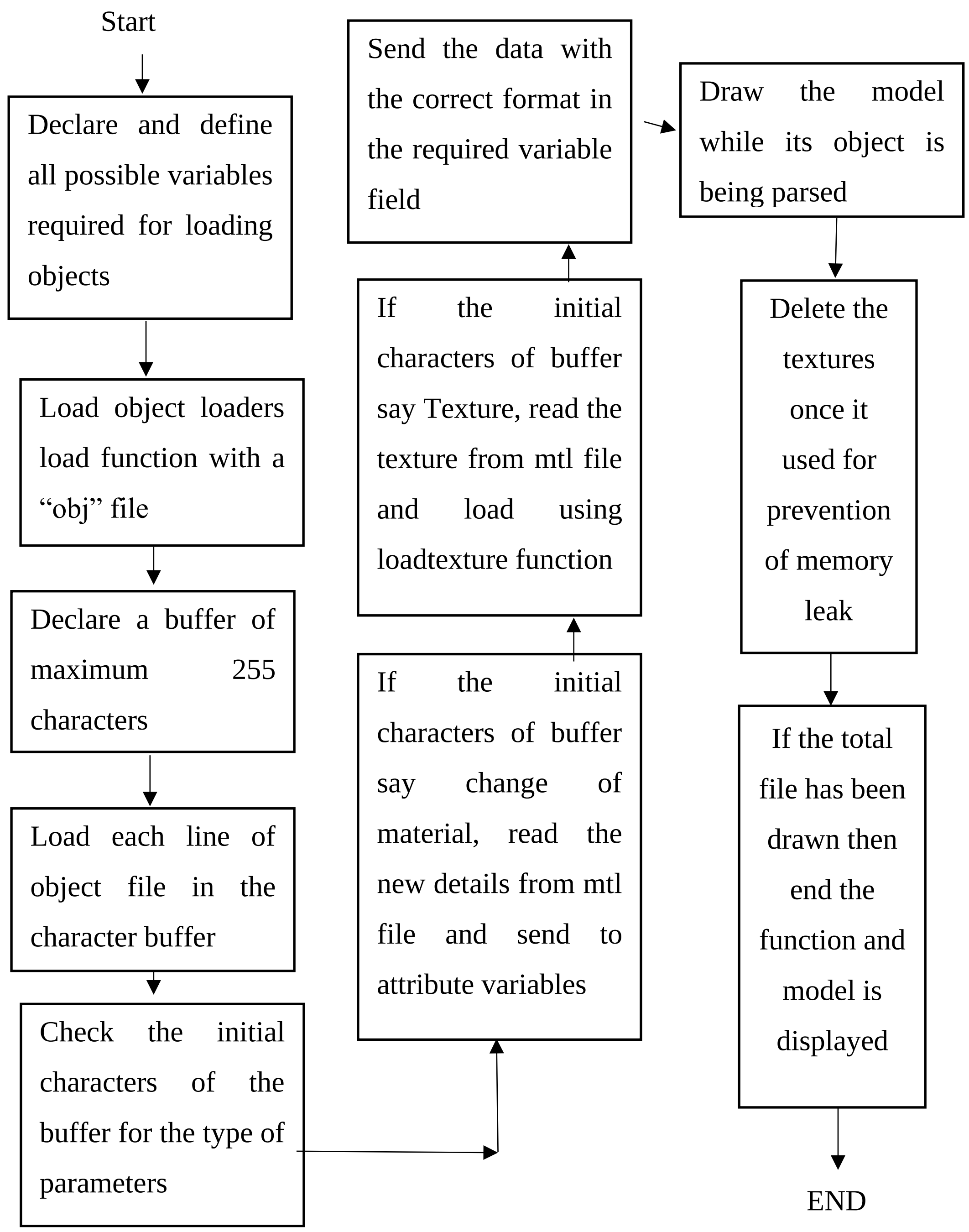

Start
Declare and define all possible variables required for loading objects
Load object loaders load function with a "obj" file
Declare a buffer of maximum 255 characters
Load each line of object file in the character buffer
Check the initial characters of the buffer for the type of parameters
If the initial characters of buffer say change of material, read the new details from mtl file and send to attribute variables
If the initial characters of buffer say Texture, read the texture from mtl file and load using loadtexture function
Send the data with the correct format in the required variable field
Draw the model while its object is being parsed
Delete the textures once it used for prevention of memory leak
If the total file has been drawn then end the function and model is displayed
END

Fig: Block Diagram to show creation of Object Loader in OpenGL

5.2.2 SKYBOX:

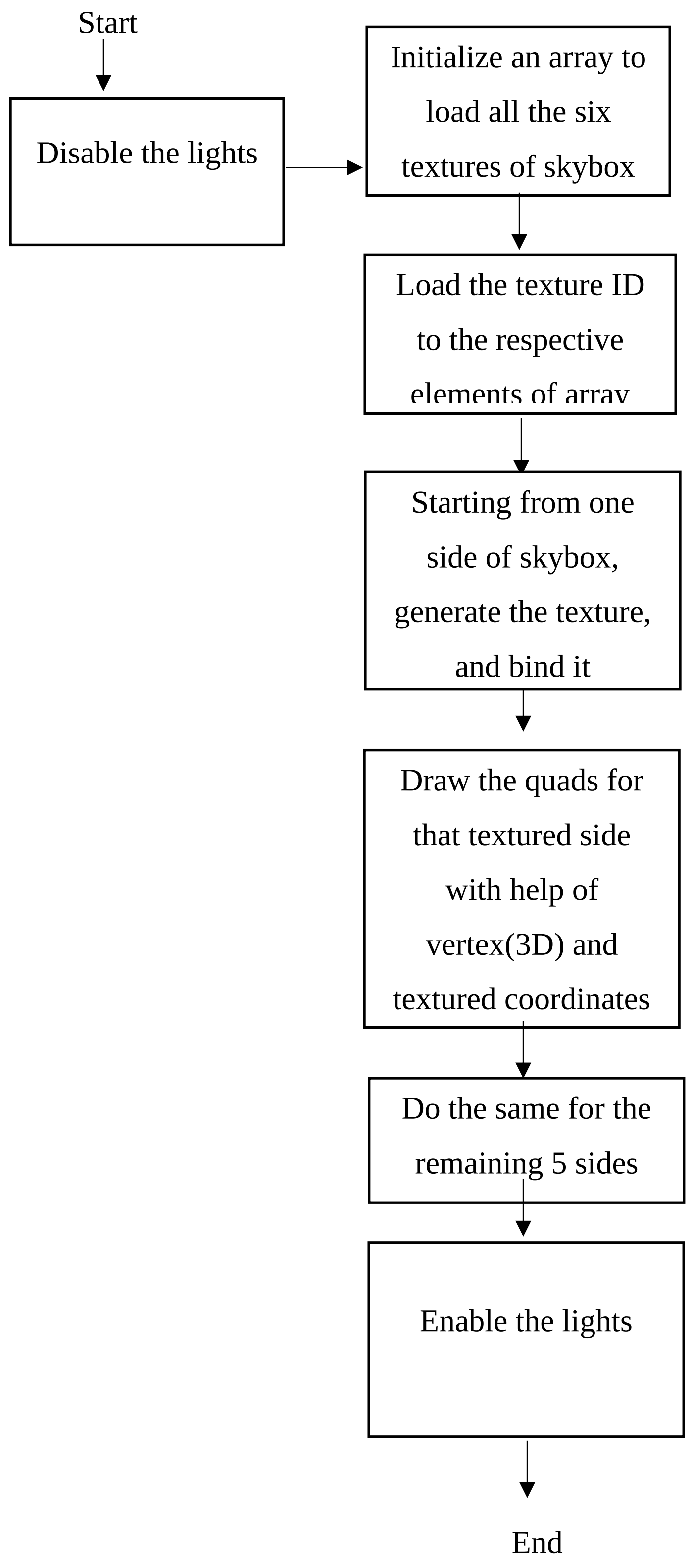
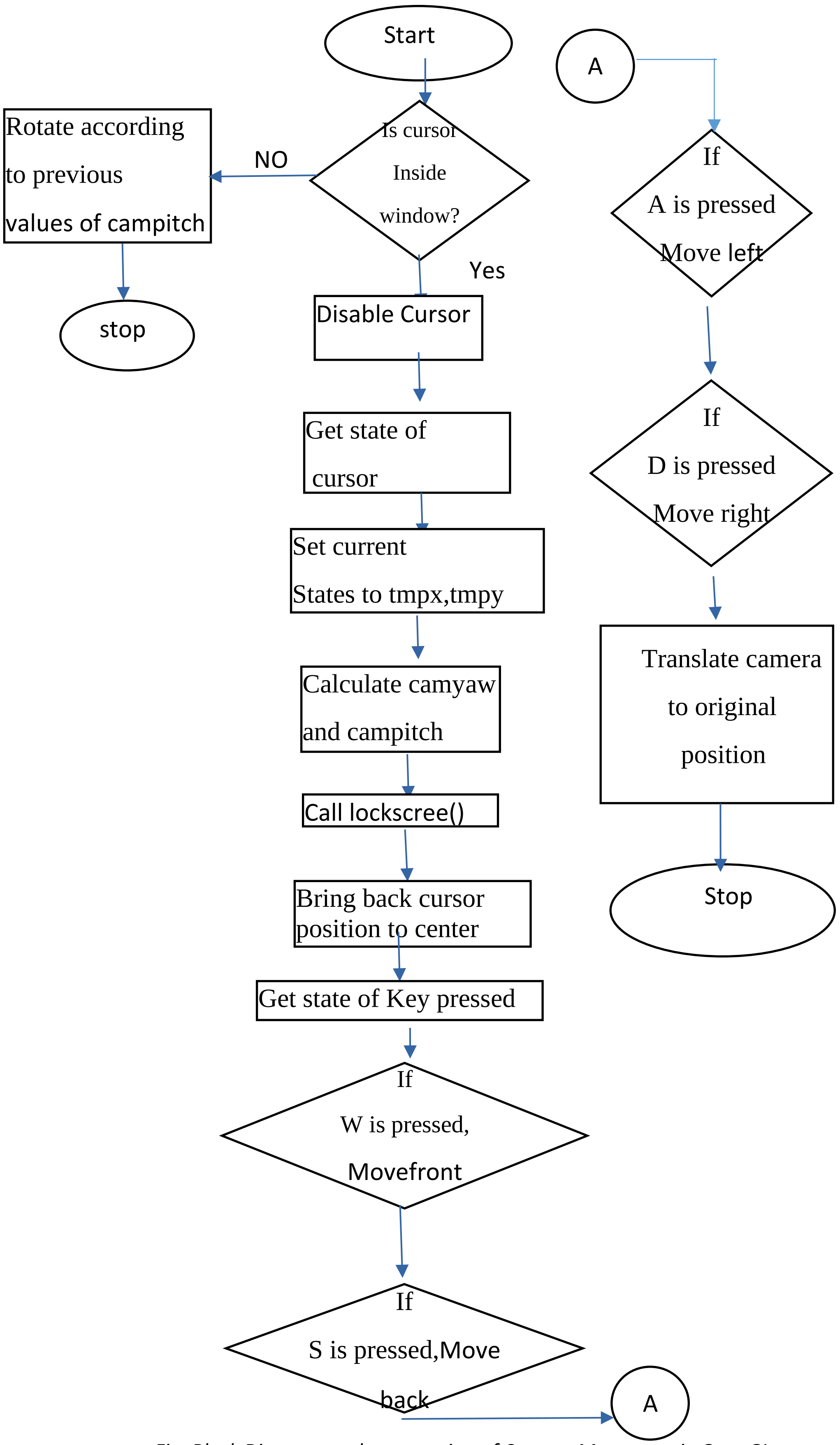

Start
Disable the lights
Initialize an array to load all the six textures of skybox
Load the texture ID to the respective elements of array
Starting from one side of skybox, generate the texture, and bind it
Draw the quads for that textured side with help of vertex(3D) and textured coordinates
Do the same for the remaining 5 sides
Enable the lights
End

Fig: Block Diagram to show creation of Skybox in OpenGL

5.2.3 CAMERA MOVEMENT:



Start
Is cursor Inside window?
NO
Rotate according to previous values of campitch
stop
Yes
Disable Cursor
Get state of cursor
Set current States to tmpx,tmpy
Calculate camyaw and campitch
Call lockscree()
Bring back cursor position to center
Get state of Key pressed
If W is pressed, Movefront
If S is pressed,Move back
A
A
If A is pressed Move left
If D is pressed Move right
Translate camera to original position
Stop

Fig: Block Diagram to show creation of Camera Movement in OpenGL

5.2.4 PARTICLE EFFECT:

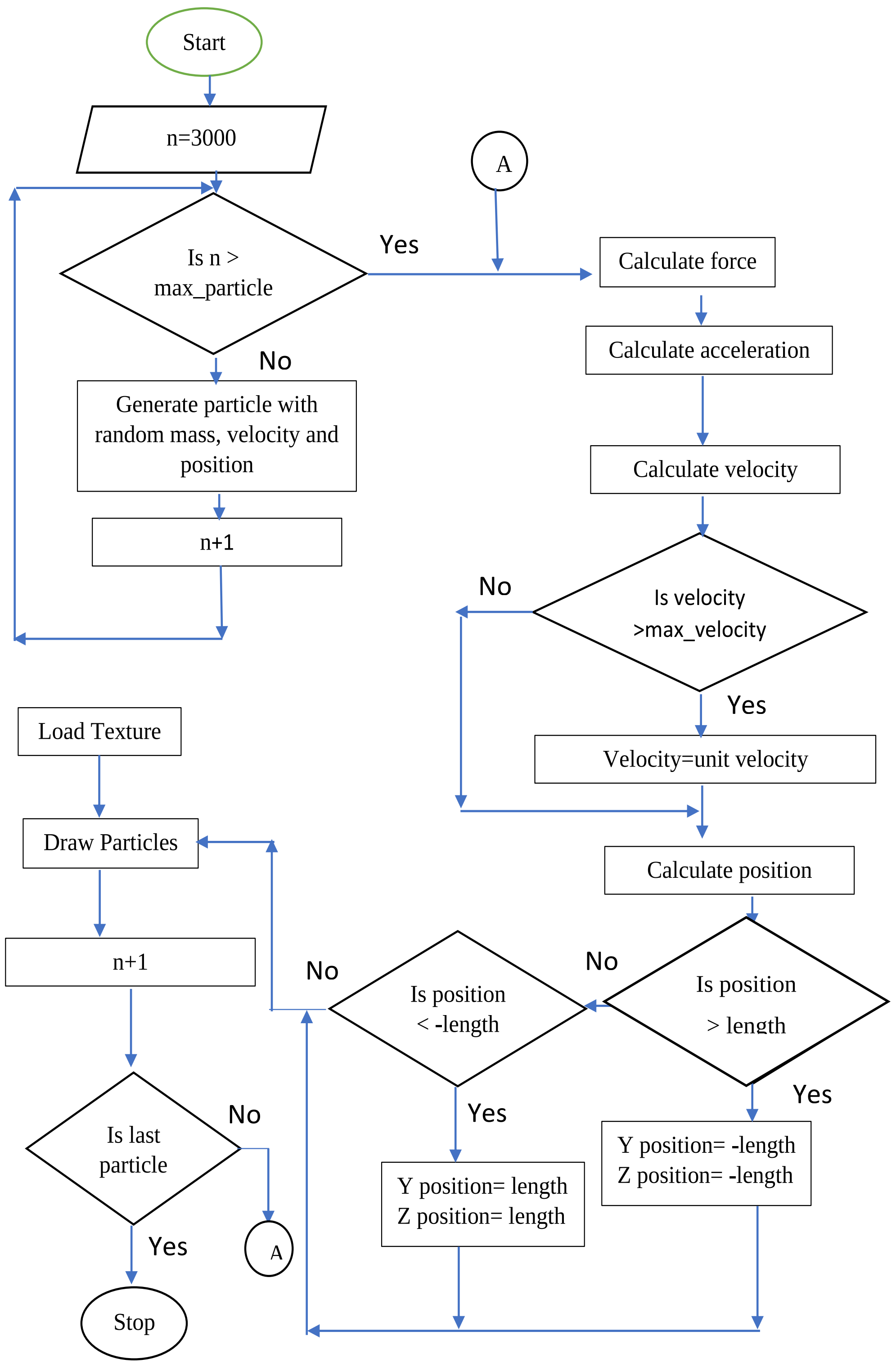


Fig: Block Diagram to show creation of Particle Effect in OpenGL

6. RESULT

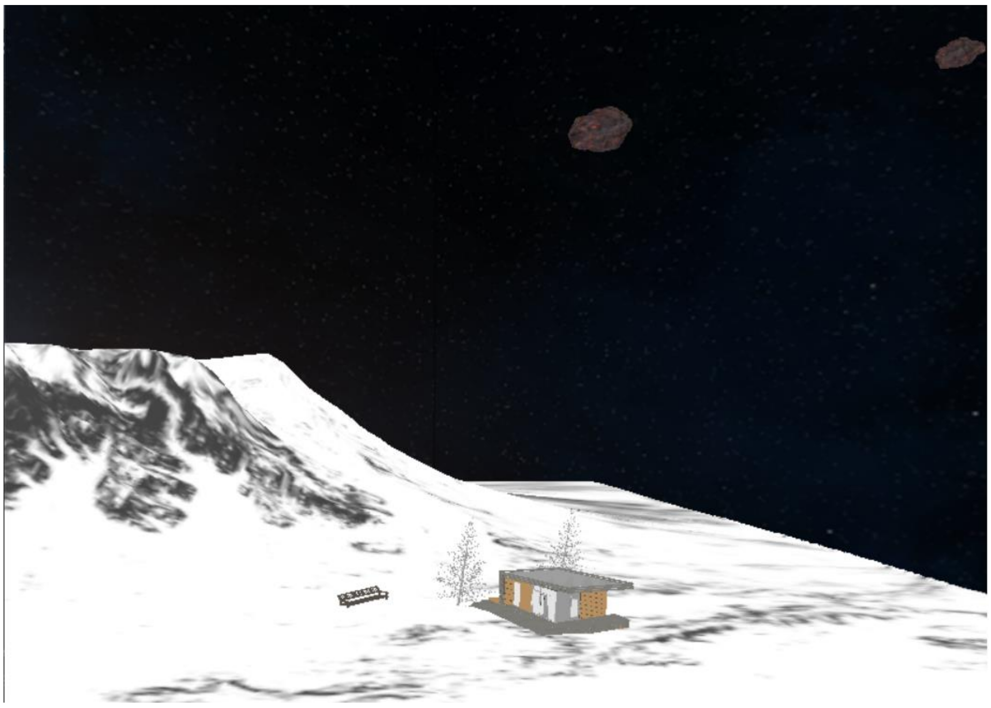


Figure 10 Object loaded with skybox

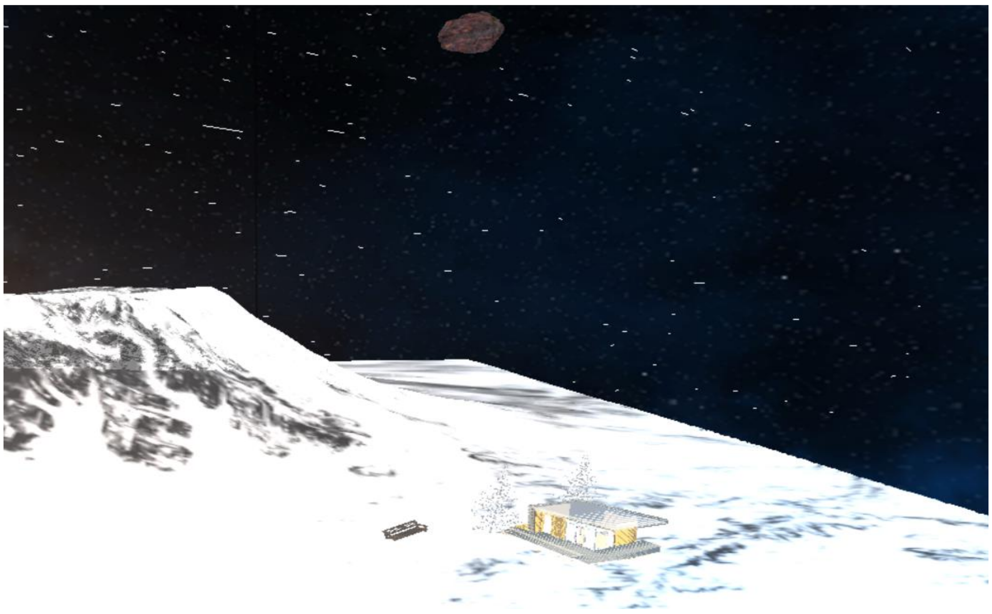


Figure 11Object loaded with skybox and particle system

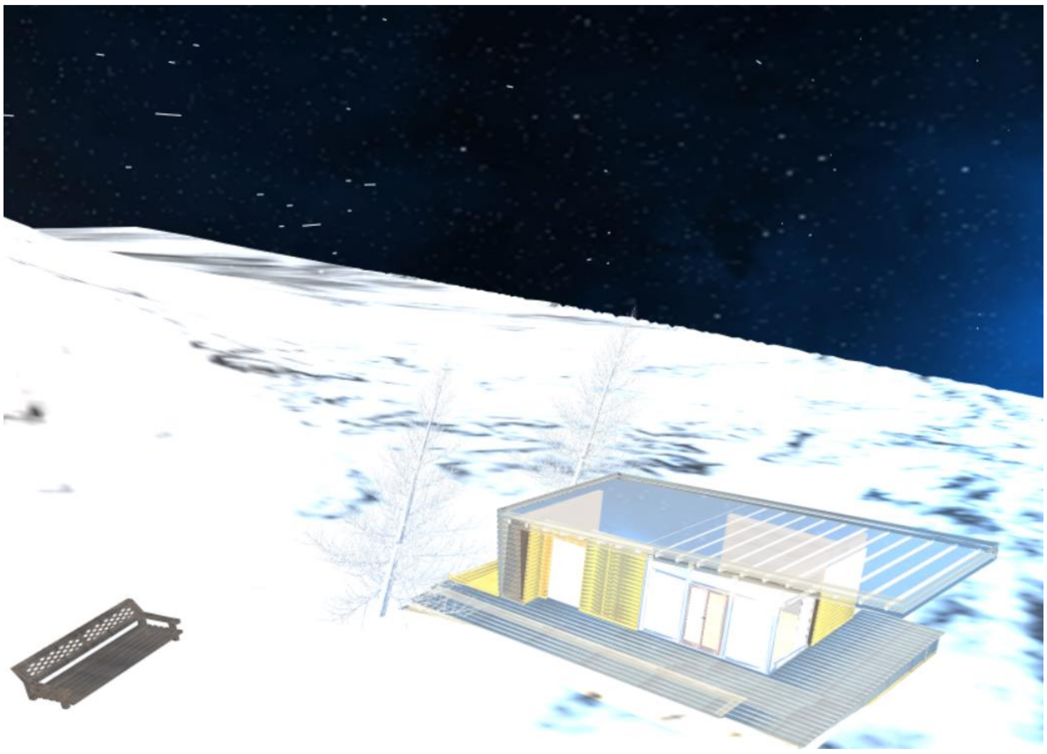


Figure 12 MultiSample AntiAliasing final result

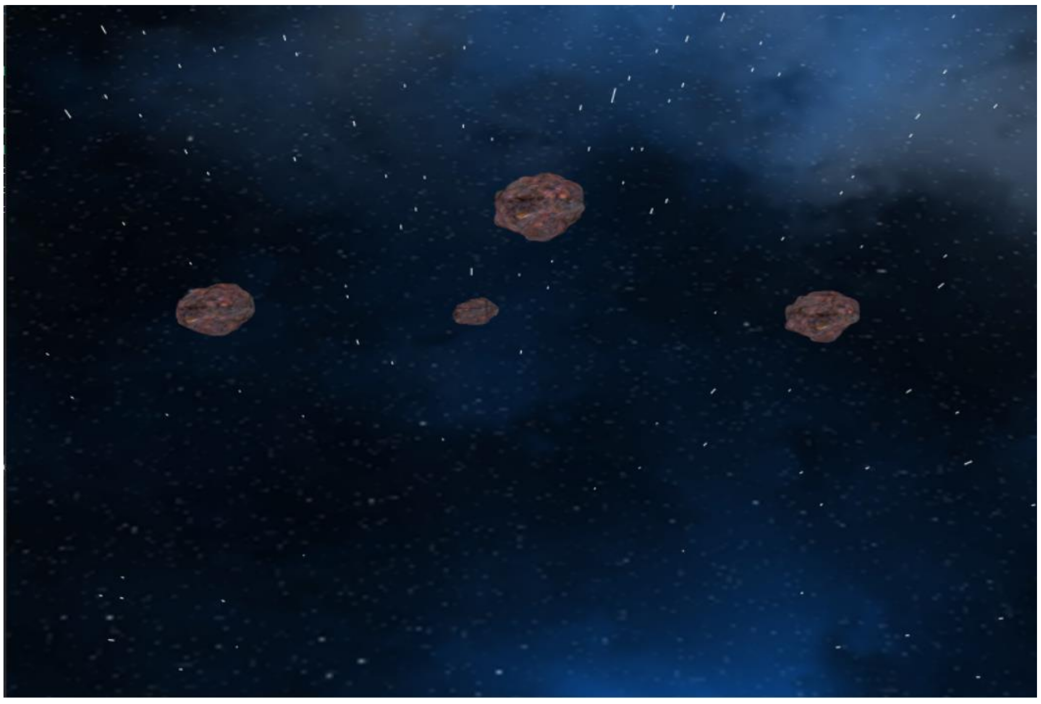


Figure 13 Meteors translated result

7. DISCUSSION

The design and rendering of the "Nightfall" using legacy OpenGL and blender was done successfully. We didn't face any major hiccups while the doing of this project. But, since we weren't familiar with the usage of OpenGL and blender as we haven't used such tools and coding in any field before, it was a bit challenging for us to get a grasp of it in earlier days of this project. Thus, we needed to consult many tutorials and guides and study them thoroughly to get some basic grasps of 3D graphics designing.

Our approach for this project was to divide and conquer. Instead of designing all of the necessary objects at once in a single device, we approached the project by breaking the project into various parts. The project was divided into object creation, designing of house, character creation, meteor creation and its movement, snow and fire effect creation and lighting. Each member handled the part of the project that was assigned to them based on their capabilities and the time that was available to them.

All of the object and character creation was performed through the use of Blender. Blender provided a simple but effective interface for object creation process. The meteor and its motion were also designed using blender while the necessary particle effects (snow) were created using particle system. While the illumination portion of the project was handled through the use of legacy OpenGL.

Though, our project was completed successfully, we could see that there were some drawbacks and limitations to how final product looked. These limitations were also because of lack of extensive knowledge and application of the knowledge that we did have into practicality due to time constraints.

7.1 Advantages:

1. Legacy opengl means it can be run in even old computers, provided resources are enough.
2. Easy to debug and reimplement because no any advanced knowledge of modern OpenGL required.

3. No need of any shader files.
4. Particle effect simulates snow/meteor strikes properly, up to a given time.
5. Skybox provides a realistic angle to the scene.

7.2 LIMITATIONS:

1. Legacy opengl meaning we couldn’t use shaders scripts and files, thus limited flexibility.
2. Performance isn’t as good as it would have been if core opengl was used.
3. Particle system is not complete because of very limited control over particles after sometime.
4. No Phong model illumination.
5. Requires very high amount of memory because of huge textures, particularly the texture of the terrain.

8. CONCLUSION

In our project “Nightfall” we have indeed successfully graphically designed and replicated the motion of a meteor in a night sky. We also added a mountain with its terrain containing objects such as house, trees, and bench to make our project look interesting. Then by combining all of these objects, we ran an animation depicting how the meteor looks in a night sky with an environment containing such objects. The animation could be viewed from any angle by changing the position of the camera through the use of directional keys and mouse/touchpad.

With the conclusion of the project, we understood the basics in computer graphics designing that will certainly be a stepping stone in our future projects in related field. We were also able to learn and implement various OpenGL techniques. We, thus, have become sufficiently proficient in field of Computer Graphics through use of 3D designing software like Blender and OpenGL.

The project was completed successfully and the required objects were designed successfully. Along with the completion of the project, we also fulfilled all of our objectives and thus the project was achieved wholly.

9. REFERENCES

- Donald D. Hearn and M. Pauline Baker (2004), "Computer Graphics with OpenGL (3rd Edition)" Pearson Publication
- Gregsidelnikov, "Legacy opengl tutorial" , *NeHe Production,* http://nehe.gamedev.net/tutorial/lessons_01__05/22004/
- Gregsidelnikov, " Particle Engine Using Triangle Strips", *NeHe Production*, http://nehe.gamedev.net/tutorial/particle_engine_using_triangle_strips/21001/
- kevindhawkins, " gamedev-net/nehe-opengl ", *GitHub,* https://github.com/gamedev-net/nehe-opengl/tree/master/devc
- Bill Jacobs, " Lesson 13: Particle System", *OpenGL tutorial* , http://www.videotutorialsrock.com/opengl_tutorial/particle_system/home.php
- Bill Jacobs, " Lesson 5: Textures", *OpenGL tutorial,* http://www.videotutorialsrock.com/opengl_tutorial/textures/home.php
- Bill Jacobs, " Lesson 6:Lighting", *OpenGL tutorial*, http://www.videotutorialsrock.com/opengl_tutorial/lighting/home.php
- "Tutorial 25 SkyBox" , *OGLdev* , http://ogldev.atspace.co.uk/www/tutorial25/tutorial25.html
- "Camera", *Learn OpenGL,* https://learnopengl.com/Getting-started/Camera
- "Tutorial 7 : Model loading" , *opengl-tutorial*, http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/
- "SDL 1.2 Tutorials", *SDL wiki,* https://wiki.libsdl.org/SDL1.2Tutorials
- Kaufmann Morgan, (2005), CHAPTER 5 - Texture Mapping, "*Advanced Graphics Programming Using OpenGL 1st Edition*", (pp. 73-102), Retrieved from http://www.r-5.org/files/books/computers/algo-list/realtime-3d/Tom_McReynolds-Advanced_Graphics_Programming-EN.pdf

10. BIBLIOGRAPHY

- “Wikipedia the free encyclopedia”, *Wikipedia*, https://www.wikipedia.org/
- *Khronos Group Connecting Software to Silicon*, https://www.khronos.org/
- Jackie Neiders, Tom Davis and Mason Woo, (February 1, 1993) “*Opengl Programming Guide: The Official Guide to Learning Opengl, Release 1*”
- Dave Shreiner , “*OpenGL(R) Reference Manual: The Official Reference Document to OpenGL, Version 1.4 (4th Edition)”*
- *Stack Overflow*, https://stackoverflow.com/
- “Google Search Engine*”, Google*, https://www.google.com/
- *GLFW*, https://www.glfw.org/
- *YouTube*, https://www.youtube.com/
- Richard S. Wright, Jr. Nicholas Haemel (2015), “*OpenGL Super bible, Seventh Edition, Comprehensive Tutorial and Reference*”, Graham Sellers