TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A PROJECT REPORT ON

COMPUTER GRAPHICS

(GraphOE)

SUBMITTED BY:

Dinesh Lamichhane (074 BEX 410)

Sandeep Poudel (074 BEX 440)

Sirish Pachhai (074 BEX 443)

Yadav Dhakal (074 BEX 448)

SUBMITTED TO:

DEPARTMENT OF COMPUTER

AND ELECTRONICS ENGINEERING

DECLARATION

We hereby declare that the work in this report is our own unaided work. It is being submitted to the Department of Computer and Electronics engineering as our Computer Graphics Project. This project report was completed by the great effort of our four team members. It has not been submitted or used for any other project submission. The work contained in the report is original and whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

ACKNOWLEDGMENT

We would like to express our deepest gratitude to **Er. Basanta Joshi,** lecturer of Computer Graphics for his encouragement and guidance in the completion of the Project. We acknowledge our Department of Computer and Electronics Engineering for including Computer Graphics Project in our curriculum as it enhanced our theoretical knowledge.

We would like to thank our lab teachers for their assistance and guidance in computer graphics lab which turns out as a great base for keeping our progress on schedule. This lab provided us with certain guidance on implementation of various graphics algorithms.

We would also like to thank our classmates and seniors, directly or indirectly involved, for their co-operation in this project. Without them, this project would not have been completed.

ABSTRACT

In our project “GraphOE” we created an web app for plotting 2D and 3D graphs. This project shows 2D and 3D graphs of given equation. We have used javascript as our programming language and webgl as our rendering graphics library. React is used as a framework for creating web application. We have added the feature of choosing perspective and orthographic projection. Various lightning effects like shininess, diffusion, etc has been applied using appropriate algorithms like phongs algorithm. The camera angle can also be changed in 3D graphs to view the graphs from different angles.

TABLE OF CONTENTS

OBJECTIVES

This project was started with certain prime objectives which are listed as follows:

1. To get familiar with 2D and 3D rendering pipeline
2. To learn practical implementation of mathematical equations in digital world
3. To learn about the programs that resides on the GPU i.e. Shaders
4. To perform different transformations like rotation about coordinate axes, translation, scaling, etc. on the object.
5. To color and create lighting effects for the objects.
6. To learn the essence and ethics of working as a team

INTRODUCTION

Computer graphics is a core technology that is widely used for generating, storing and manipulating images. Digital photography, film, video games, cell phone and computer displays, etc are many specialized application of computer graphics. A great deal of specialized hardware and software has been developed, with the displays of most devices being driven by computer graphics hardware.

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.

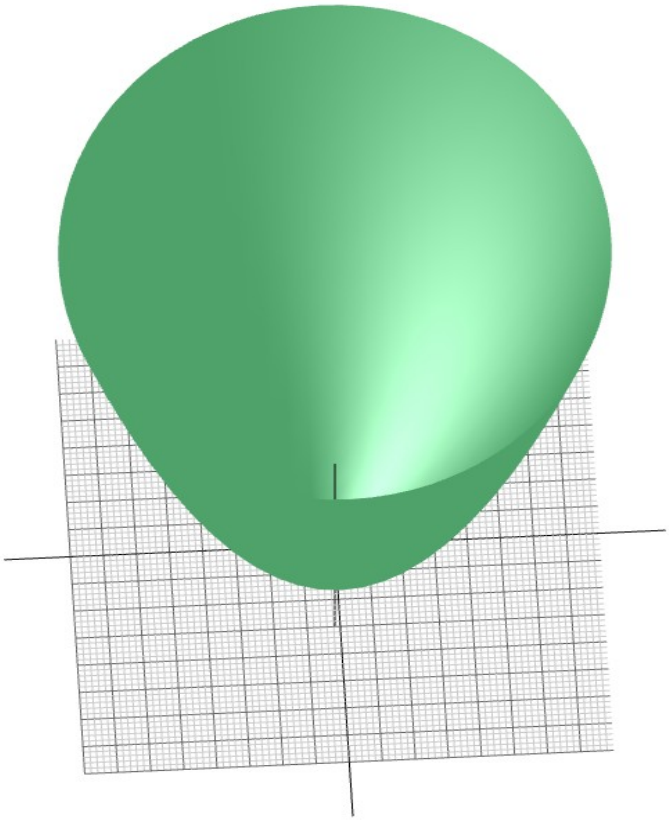
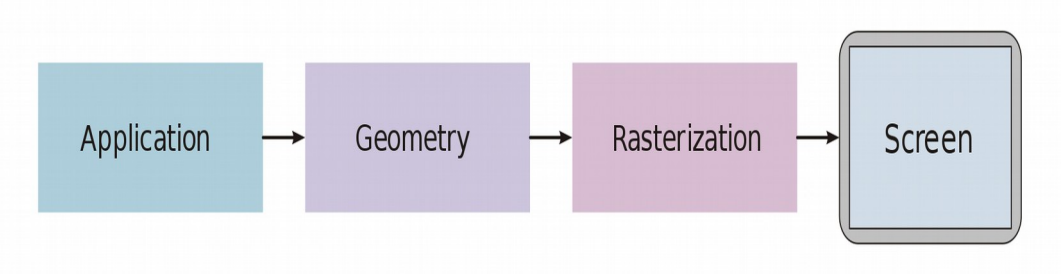


Fig: Paraboloid Using Computer Graphics

THEORY

Graphics Pipeline:

In computer graphics, a computer graphics pipeline is a conceptual model that describes what steps a graphics system needs to perform to render a 3D scene to a 2D screen. Once a 3D model has been created, for instance in a video game or any other 3D computer animation, the graphics pipeline is the process of turning that 3D model into what the computer displays. Because the steps required for this operation depend on the software and hardware used and the desired display characteristics, there is no universal graphics pipeline suitable for all cases.



Application

The application step is executed by the software on the main processor (CPU). In the application step, changes are made to the scene as required, for example, by user interaction by means of input devices or during an animation. In a modern Game Engine such as Unity, the programmer deals almost exclusively with the application step, and uses a high-level language such as C#, as opposed to C or C++. The new scene with all its primitives, usually triangles, lines and points, is then passed on to the next step in the pipeline.

Geometry

Modelling Transformation and Viewing Transformation can be done by 3D transformations. The viewing-coordinate system is used in graphics packages as a reference for specifying the observer viewing position and the position of the

projection plane. Projection operations convert the viewing-coordinate description (3D) to coordinate positions on the projection plane (2D). (Usually combined with clipping, visual-surface identification, and surface-rendering)Workstation transformation maps the coordinate positions on the projection plane to the output device

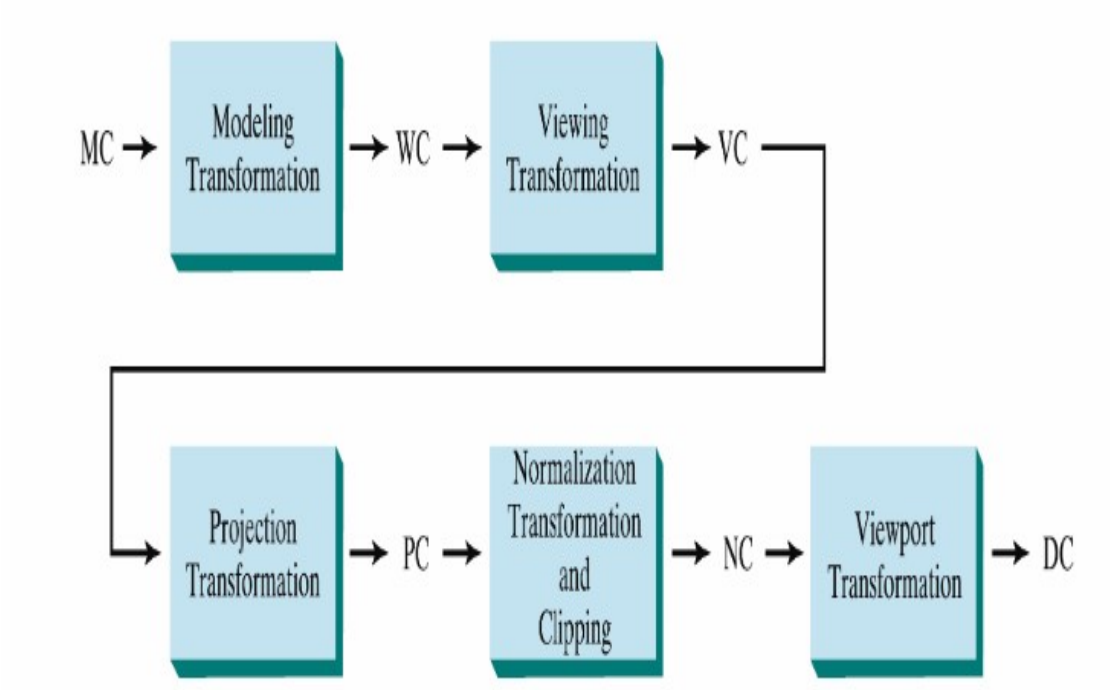


Fig: 3D viewing pipeline

Rasterization

Rasterization is the process by which a primitive is converted to a two-dimensional image. Each point of this image contains such information as color and depth. Thus, rasterizing a primitive consists of two parts. The first is to determine which squares of an integer grid in window coordinates are occupied by the primitive. The second is assigning a color and a depth value to each such square."

Rasterization is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (a series of pixels, dots or lines, which, when displayed together, create the image which was represented via

shapes). The rasterized image may then be displayed on a computer display, video display or printer, or stored in a bitmap file format. Rasterization may refer to either the conversion of models into raster files, or the conversion of 2D rendering primitives such as polygons or line segments into a rasterized format.

Rasterization is the process by which most modern display systems turn electronic data or signals into projected images, such as video or still graphics. This is typically a process of identifying the needs of a specific media configuration, then allocating resources so that images are efficiently and optimally projected on the display device.

Rasterized graphics are often compared with image vectors. While rasterization is typically a process of compiling scan lines or pixels on a bitmap, in contrast, vectors incorporate mathematical functions in order to create images based on geometric shapes, angles and curves.

Rasterisation of 3D images

Compared with other rendering techniques such as ray tracing, rasterisation is extremely fast. However, rasterisation is simply the process of computing the mapping from scene geometry to pixels and does not prescribe a particular way to compute the color of those pixels. Shading, including programmable shading, may be based on physical light transport, or artistic intent.

The process of rasterising 3D models onto a 2D plane for display on a computer screen ("screen space") is often carried out by fixed function hardware within the graphics pipeline. This is because there is no motivation for modifying the techniques for rasterisation used at render time[clarification needed] and a special-purpose system allows for high efficiency.

Triangle rasterization

A common representation of digital 3D models is polygonal. Before rasterization, individual polygons are broken down into triangles, therefore a typical problem to solve in 3D rasterization is rasterization of a triangle. Properties that are usually required from triangle rasterization algorithms are that rasterizing two adjacent triangles (i.e. those that share an edge)

1. leaves no holes (non-rasterized pixels) between the triangles, so that the rasterized area is completely filled (just as the surface of adjacent triangles). And
2. no pixel is rasterized more than once, i.e. the rasterized triangles don't overlap. This is to guarantee that the result doesn't depend on the order in which the triangles are rasterized. Overdrawing pixels can also mean wasting computing power on pixels that would be overwritten.

This leads to establishing rasterization rules to guarantee the above conditions. One set of such rules is called a top-left rule, which states that a pixel is rasterized if and only if

1. its center lies completely inside the triangle, or
2. its center lies exactly on the triangle edge (or multiple edges in case of corners) that is (or, in the case of corners, all are) either top or left edge.

A top edge is an edge that is exactly horizontal and lies above other edges, and a left edge is a non-horizontal edge that is on the left side of the triangle.

LITERATURE REVIEW

By surveying the web and going through different articles, tutorials and documentation, we acquired enough knowledge of rendering pipeline and WebGL functions, then finally used it for our project development. Tutorials and explanations given on the website learnwebglfundamentals.com were extremely helpful for our project which provides the best reference and workflow for our project.

We referred to the various algorithms like point and tracing for the curve generation. We also have to understand various concept of projection, lightning, camera angle, etc. This project also involved react native as framework and javascript as programming language. So, we also went through the documentation to understand the fundamental comcept.

METHODOLOGY

Webgl

WebGL (Web Graphics Library) is a JavaScript API for rendering high-performance interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins. WebGL does so by introducing an API that closely conforms to OpenGL ES 2.0 that can be used in HTML5 <canvas> elements. This conformance makes it possible for the API to take advantage of hardware graphics acceleration provided by the user's device.

Support for WebGL is present in Firefox 4+, Google Chrome 9+, Opera 12+, Safari 5.1+, Internet Explorer 11+, and Microsoft Edge build 10240+; however, the user's device must also have hardware that supports these features.

The WebGL 2 API introduces support for much of the OpenGL ES 3.0 feature set; it's provided through the WebGL2RenderingContext interface.

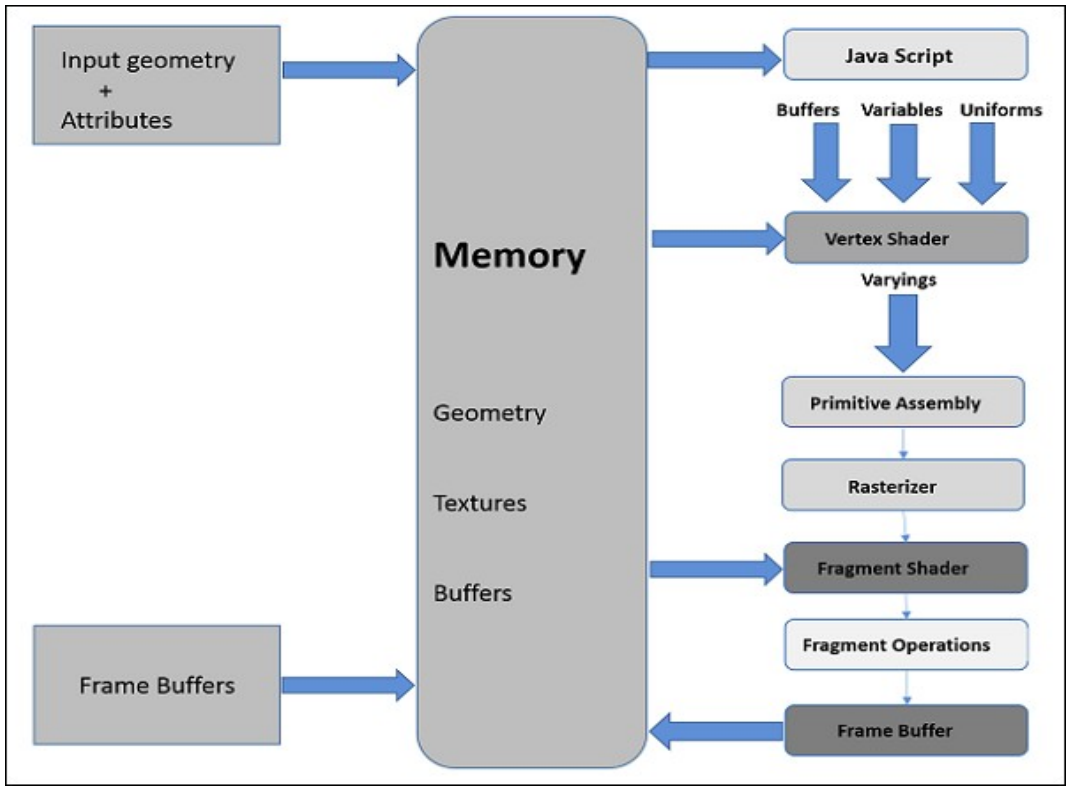


Fig: Webgl Rendering Pipeline

Shaders

A shader is a program, written using the OpenGL ES Shading Language (GLSL), that takes information about the vertices that make up a shape and generates the data needed to render the pixels onto the screen: namely, the positions of the pixels and their colors. There are two shader functions run when drawing WebGL content: the vertex shader and the fragment shader. You write these in GLSL and pass the text of the code into WebGL to be compiled for execution on the GPU. Together, a set of vertex and fragment shaders is called a shader program.

Vertex shader

Each time a shape is rendered, the vertex shader is run for each vertex in the shape. Its job is to transform the input vertex from its original coordinate system into the clipspace coordinate system used by WebGL, in which each axis has a range from -1.0 to 1.0, regardless of aspect ratio, actual size, or any other factors.
The vertex shader must perform the needed transforms on the vertex's position, make any other adjustments or calculations it needs to make on a per-vertex basis, then return the transformed vertex by saving it in a special variable provided by GLSL, called gl_Position.The vertex shader can, as needed, also do things like determine the coordinates within the face's texture of the texel to apply to the vertex, apply the normals to determine the lighting factor to apply to the vertex, and so on. This information can then be stored in varyings or attributes as appropriate to be shared with the fragment shader.

Fragment shader

The fragment shader is called once for every pixel on each shape to be drawn, after the shape's vertices have been processed by the vertex shader. Its job is to determine the color of that pixel by figuring out which texel (that is, the pixel from within the shape's texture) to apply to the pixel, getting that texel's color, then applying the appropriate lighting to the color. The color is then returned to the WebGL layer by storing it in the special variable gl_FragColor. That color is

then drawn to the screen in the correct position for the shape's corresponding pixel.

Algorithms

Visualization of an implicit curve

To visualize an implicit curve one usually determines a polygon on the curve and displays the polygon. For a parametric curve this is an easy task: One just computes the points of a sequence of parametric values. For an implicit curve one has to solve two subproblems:

1. determination of a first curve point to a given starting point in the vicinity of the curve,
2. determination of a curve point starting from a known curve point.

In both cases it is reasonable to assume grad F ≠ (0 , 0). In practice this assumption is violated at single isolated points only.

Point algorithm

For the solution of both tasks mentioned above it is essential to have a computer program which we will call (C P o i n t) , which, when given a point Qo= (Xo , Yo) near an implicit curve, finds a point P that is exactly on the curve:

1. (P1) for the start point is j=0
2. (P2) repeat

$$(x_{j+1}, y_{j+1}) = (x_j, y_j) - \frac{F(x_j, y_j)}{F_x(x_j, y_j)^2 + F_y(x_j, y_j)^2} \, (F_x(x_j, y_j), F_y(x_j, y_j))$$

Newton step for function $g(t) = F\left(x_j + tF_x(x_j, y_j), y_j + tF_y(x_j, y_j)\right)$.

3.(P3) until the distance between the points . is small enough.

4.(P4) . is the curve point near the start point ..

Tracing Algorithm

In order to generate a nearly equally spaced polygon on the implicit curve one chooses a step length s and

1. (T1) chooses a suitable starting point in the vicinity of the curve.
2. (T2) determines a first curve point P1 using program Cpoint.

3. (T3) determines the tangent, chooses a starting point on the tangent using step length and determines a second curve point P2 using program Cpoint.

Because the algorithm traces the implicit curve it is called a tracing algorithm. The algorithm traces only connected parts of the curve. If the implicit curve consists of several parts it has to be started several times with suitable starting points.

3D Projection

A 3D projection or graphical projection maps points in three-dimensions onto a two-dimensional plane. As graphics are usually displayed on two-dimensional media such as paper and computer monitors, these projections are widely used, especially in engineering drawing, drafting, and computer graphics.

Orthographic projection

The orthographic projection is derived from the principles of descriptive geometry and is a two-dimensional representation of a three-dimensional object. It is a parallel projection (the lines of projection are parallel both in reality and in the projection plane). It is the projection type of choice for working drawings.

If the normal of the viewing plane (the camera direction) is parallel to one of the primary axes (which is the *x*, *y*, or *z* axis), the mathematical transformation is as follows; To project the 3D point ax, ay, az onto the 2D point bx, by using an orthographic projection parallel to the y axis (where positive *y* represents forward direction - profile view), the following equations can be used:

$$b_x = s_x a_x + c_x$$

$$b_y = s_z a_z + c_z$$

$$\begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + \begin{bmatrix} c_x \\ c_z \end{bmatrix}.$$

Perspective projection

Perspective projection is a linear projection where three dimensional objects are projected on a picture plane. This has the effect that distant objects appear smaller than nearer objects.

It also means that lines which are parallel in nature (that is, meet at the point at infinity) appear to intersect in the projected image, for example if railways are pictured with perspective projection, they appear to converge towards a single point, called the vanishing point. Photographic lenses and the human eye work in the same way, therefore perspective projection looks most realistic.Perspective projection is usually categorized into one-point, two-point and three-point perspective, depending on the orientation of the projection plane towards the axes of the depicted object.

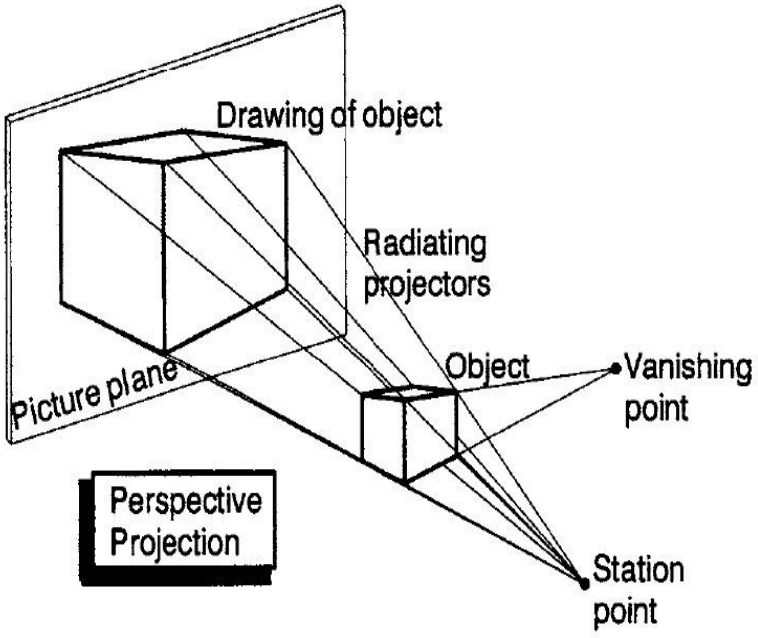
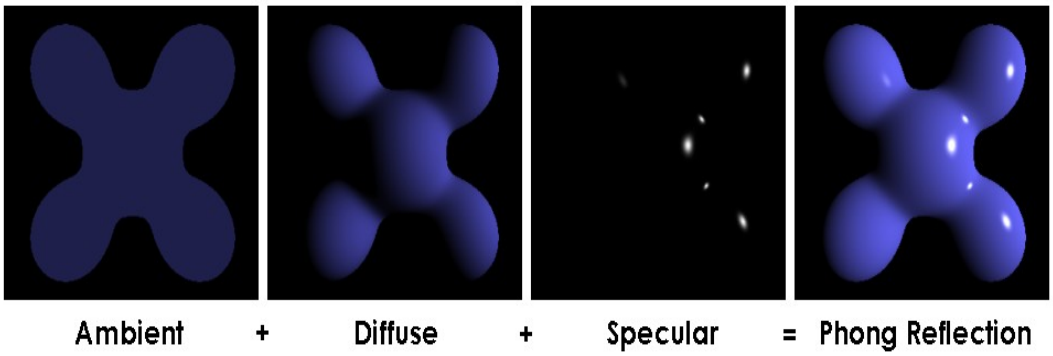


Figure 2.2 *Perspective projection*

Phongs Reflection Model

The Phong reflection model is an empirical model of the local illumination of points on a surface. In 3D computer graphics, it is sometimes referred to as "Phong shading", in particular if the model is used with the interpolation method of the same name and in the context of pixel shaders or other places where a lighting calculation can be referred to as “shading”.

Phong reflection is an empirical model of local illumination. It describes the way a surface reflects light as a combination of the diffuse reflection of rough surfaces with the specular reflection of shiny surfaces. It is based on Phong's informal observation that shiny surfaces have small intense specular highlights, while dull surfaces have large highlights that fall off more gradually. The model also includes an ambient term to account for the small amount of light that is scattered about the entire scene.



Then the Phong reflection model provides an equation for computing the illumination of each surface point Ip:

$$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \in \text{ lights}} (k_\mathrm{d} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}}).$$

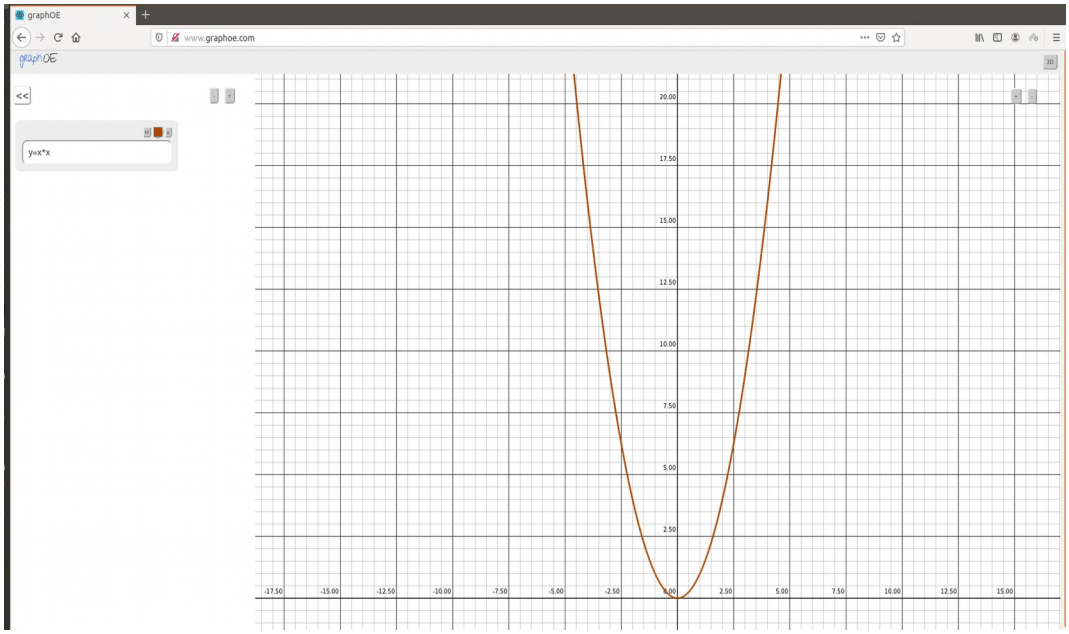
IMPLEMENTATION

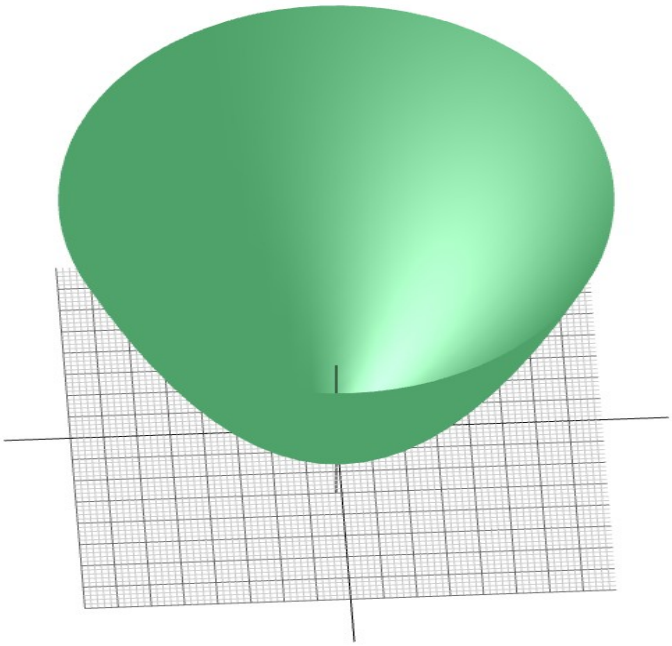
As discussed earlier all the rendering code is written in javascript programming language using WebGL library. We use two shaders: vertex shader and fragment shader. Vertex shader stores all the textures coordinates,normals in certain memory locations. These vertices are passed to fragment shader as well. Light source, direction of light and its intensity are handled by fragment shader. Based on types of light source,its intensity, distance of object from light source and material body of textures, its lighting effect is determined.

Various algorithms were used to drew graphs of given function. We used point and tracing algorithm to draw the graphs of implicit function. We used the concept of lightning using phongs reflection model.

Various mathematical concepts are necessary for rendering. All this is provided by the WebGL mathematics library. Mainly 3D coordinates are projected to 2D coordinates and rendered on our 2D screen. Matrix operation is very essence which is used for translation, scaling, rotation and other calculations.

OUTPUT SCREENSHOTS





PROBLEM FACED

During the development process of 2D and 3D graph plotter we faced following problems:

1.It was difficult in implementing various implicit functions algorithms.

2.We were not very much familiar with the GPU programs

3.Proper documentation of webgl wasn't available.

4. It was difficult for generating smooth 3D curve of certain given functions.

LIMITATIONS AND FUTURE ENHANCEMENT

Due to time limitations and WebGL being a hard topic there are some limitations in our project. We can extend its features and make it even better. As WebGL is only supported in Firefox 4+, Google Chrome 9+, Opera 12+, Safari 5.1+, Internet Explorer 11+, and Microsoft Edge graph doesn't render properly in other lesser versions of browser. The UI of our web app is still under development. We are planning to continue with that. Also, we are planning to implement features of user authentication and graph saving features for user in near future.

CONCLUSIONS AND RECOMMENDATIONS

Hence, “GraphOE”, a web application is created in javascript using webgl as graphics library and react as framework for web in Visual Studio. If anyone has a great desire to know the logic and mathematics behind 3D Graphics pipeline, WebGL is best to learn for making web application. It makes everyone familiar with GPU programming and helps to understand everything from scratch.

REFERENCES

1. John F. Hughes ,Andrises Van Dam , David F. Skylar “Computer Graphics Principles and Pratise”
2. Foley, Van Dam, Feiner, Hughes “Computer Graphics principles and practice (Second Edition in C”)
3. https://webglfundamentals.org, MDN WebGL