Tribhuwan University

Institute Of Engineering

Pulchowk Campus

Lalitpur, Nepal

A Project Report On

Virtual Tour of Temple

Submitted By:

Ayush Shrestha (068/BEX/408)

Bidhan Sthapit (068/BEX/411)

Jamuna Maharjan (068/BEX/419)

A project Submitted in partial fulfillment of EX 603: Computer Graphics

Submitted To:

Department of Electronics and Computer Engineering

March 25, 2014

ACKNOWLEDGEMENT

First of all, we would like to express our sincere gratitude to the Department of Electronics and Computer Engineering for providing us such an opportunity and motivation for the project on Computer Graphics

.

We are indebted to our Lecturer **Basanta Joshi, PhD** for his valuable suggestions and guidelines for the project. We would also like to express our sincere gratitude to ours friends and seniors. We are also very thankful to all the persons who helped us till the completion of the project.

ABSTRACT

This report is written to be submitted to the Department of Electronics and Computer Engineering, Pulchowk Campus under the course 'Computer Graphics' (EX603) in 5th semester of Bachelor of Electronics and Communication Engineering. The target of the project is to implement the basic techniques and algorithms used in modeling and transformation of a 3D object. In this project, we have implemented 3D projection along with scanfill. We have also implemented Visual Surface Detection and Collision. During rendering, we have used Phong's illumination to model the lighting effects. Hence, basic virtual tour of temple was completed.

Table of Contents: **Page No:**

INTRODUCTION

Computer Graphics are graphics created using computers and the representation of image data by a computer specifically with help from specialized graphic hardware and software.

The interaction and understanding of computers and interpretation of data has been made easier because of computer graphics. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies and the video game industry.

Our project "**Temple Virtual Tour**" is a simulation of a historical temple environment. The temple environment consists of a two-storied temple, a bell near the temple, temple stairs. The 3D models of each are created and placed in 3D space and they will be projected in 2D plane. The type of projection used is **perspective**. The user can use keyboard to move around the temple environment. For generating **virtual realism** we have applied colouring, lighting (single source) and rendering of 3D models.

The project is about creating environment of temple. The surrounding has a main entrance from where the tour starts. The user can have the whole outer peripheral view before entering the temple. After going inside the temple through the front opening the user can have an inside view of the Pyramid. The viewer can view through different angles using the keyboard.

OBJECTIVES

The main objective of this project is to become familiar with creating 3D and 2D graphics objects in computer.

Creating graphics in computer require various graphics routines to be performed and our objective is to gain knowledge in using such routines and to list out, they are:

- To draw 3D models using **basic output primitives** like Cuboids, Pyramidal, Cylinders, Cones, Lines.
- To implement the different colours in 3D objects using various fill algorithms
- To acquire knowledge in 2-D and 3-D Geometric Transformation including basic transformations such as translation, rotation, scaling etc.
- To have concepts on 3 - dimensional Display Methods in Perspective projection.
- To know about 3 - dimensional object representation using polygon tables including vertex table, edge table and surface table.
- To apply algorithms for visible surface detection and hidden line detection(Z – buffer).
- To apply algorithms on Illumination Models and Surface Rendering Methods such as Ambient light, Diffuse Reflection, Specular Reflection and Polygon rendering methods as Gouraud Intensity Shading.
- To generate virtual reality system implementing different 2-D and 3-D algorithms.

Methodology

I. For our project, we used codeblocks IDE
II. We used OpenGL graphics library to plot the pixels.
III. We used OpenGL glut library to handle the keyboard operation.
IV. STL vector was used instead of arrays for efficient use.
V. For image loading, library was imported.
VI. Phong Illumination model for lightning and gourad shading

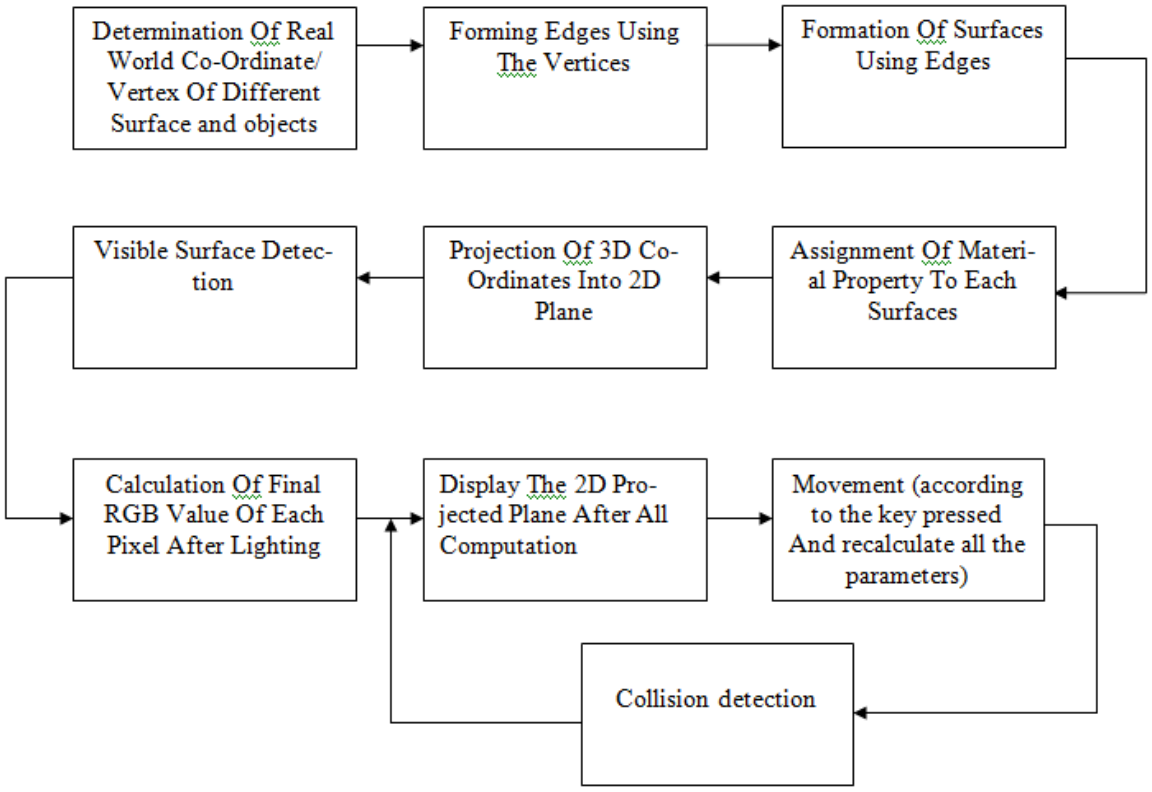

Fig. System Flow

IMPLEMENTED SYSTEM

The concepts used in the project are 3D projection, hidden surface removal, lighting and collision detection. Firstly by the help of real world points or the vertices, we created real world models. These models are developed by the help of small units of surfaces. Once the 3D objects are formed then they are projected on the projection plane. We have used the perspective projection method.

We have used numbers of classes in order to create various 3D models. By the help of parameters required to form its surfaces, each vertices of its boundary line is calculated using the appropriate formula then by the help of quardilateralization or triangularization we created the overall surfaces of the 3D models. To store the thus small surfaces we have used the STL vector container instead of the array. Then, to each surfaces we have assigned its material property then the thus obtained 3D object is projected on the projection plan. The algorithm for visible surface detection and the hidden line detection (Z-buffer) is used and the final RGB values of each pixel are calculated and finally the rendering is carried out.

For projection, a line is drawn between eye and the point in 3d space. Projection plane is defined perpendicular to the line between eye and look at coordinates at a fixed distance from the eye. Then, the point on the projection plane at which the line between eye and point intersects is the projected point on projection plane. Then the projection plane is rotated to make it perpendicular to the z-axis. Then according to the direction cosine of the viewing line, the points on projection plane are plotted on the display device (window) either from 0 to screen width or from screen width to 0.

For filling, every point on the 3d surface is calculated using algorithm similar to DDA line drawing algorithm. Then every point is given a color value (RGB) provided either manually, or by loading color values from an image.

All the points are not visible to the eye. So, if any pixel on the screen is not filled, the pixel is given color values of the 3d point projected. Then the distance between projected point on projection plane and the 3d point is calculated and stored. If another point also gets projected to the same point of projection plane, the distance calculated above is checked. If the new distance is smaller than previously calculated distance, the pixel properties are replaced by the properties of the new point projected.

For lighting, we have used Phong's illumination model. Each light and material has ambient, diffuse and specular properties. First of all, the light source position is set and all the properties of light are initialized. Then, attenuation of each light is also initialized. Then,we calculate the required angle between the surface normal and the light vector to the particular point on the very surface for the diffuse part. We also calculate the required angle between

the surface normal and the half way vector between eye and light for the specular part. Finally, using angles, material properties and light properties , we can get the final R,G,B value of the required point.

Since the users can move around and inside the temple by the help of keyboard keys, we have detected the collision between eye position and the surfaces. This is all about the 3D geometric. Firstly the point that changes by pressing of keyboards and the previous point before pressing the keyboard are checked if they lie at two sides of the projection plane or not and if the projection plane lies in between these points then there is possible of collision between the surfaces as we move. The intersection point of the line joining two points and the surface plane is found out using mathematical relations and checked if that plane lies within the boundary of that surface. If it lies inside the boundary then the previous eye position is assigned to the current eye position so that we cannot move beyond that surface.

IMPLEMENTATION

Development of Project

Given below is the development of header file global

global.h

```
const unsigned int SCREENWIDTH = 800;
const unsigned int SCREENHEIGHT = 600;
const unsigned int SCREENCENTRE[2] = {SCREENWIDTH / 2, SCREENHEIGHT/2};

    namespace ViewingSystem {
    extern float eyeCoordinates[3];
    extern float lookAtCoordinates[3];

    extern float previousFirstPointOnProjectionPlane[3];
    extern float preserveEyeCoordinates[3];
    extern float preserveLookAtCoordinates[3];


    extern float viewingAngleWithAxes[3];//angle made by the line between eye and lookAt with axes;0 for x axis, 1 for y
                                         //axis and 2 for z axis

    extern float projectionPlaneConstants[4]; //a, b, c, and d on ax + by + cz +d = 0
                                              //Is Perpendicular to the line joining eye and lookat point
    extern float eyeDistanceFromProjectionPlane;
    extern float lengthBetweenEyeAndLookAt;
    extern float lookAtDistanceFromProjectionPlane;
    extern float firstPointOnProjectionPlane[3]; //point that should be at the centre of screen.
                                                 //Calculated as intersection between projection plane and line joining
                                                 // eye and lookat point.

    extern float directionCosine[3];
    extern float perpendicularToDirectionCosine[3];
```

```
    extern float sumOfSquareOfABC;

    extern signed char sideOfEyeFromProjectionPlane; //is -1 or 1 as of which side the eye is of the projection plane. is the
                                                     // sign of value of equation of projection plane when eye is
                                                     // substituted in the x, y and z of equation
}

extern int stepChangeOnMovement; //value to be added to or subtracted from eye and lookAt when direction keys are
pressed
extern float stepAngleOnDirectionChange; // angle to be changed when viewing direction is changed

using namespace ViewingSystem;

void calculateViewingSystemParameters();
void setEyeCoordinates();
void setEyeCoordinates(float x, float y, float z);
void setLookAtCoordinates();
void setLookAtCoordinates(float x, float y, float z);
int round(float a); //rounds a and returns the integer
void gaussElimination(vector<vector<float> > matrixToSolve, vector<float>& solutionMatrix); //Solve the matrix and
                                                    // store the solution in solutionMatrix.n is the number of unknown
                                                    // variable

template<typename T> signed char getSign(T x); //returns 1 if x is +ve and -1 if x is -ve; returns 0 if x is 0

class RealWorldObjectPoint;
class Edge;
class Surface;
class Cuboid;
class ScreenPixel;
class Cylinder;
class horizontalCylinder;
class Cone;
```

This header file contains all the global variables and functions. Namespace ViewingSystem contains all the parameters and functions necessary for viewing (eye, look at, projection plane). This header file also contains general functions like round(), getSign() and gaussElimination(). Forward declaration of all the classes is also performed in this header.

realWorldObjectPoint.h

```
class RealWorldObjectPoint {

   friend class Edge;
   friend class Surface;
   friend class Cylinder;
   friend class horizontalCylinder;
   friend class Cone;

   public:
      RealWorldObjectPoint(float x = 0, float y = 0, float z = 0);
```

```
    RealWorldObjectPoint(float point[]);
    void calculateParametersWithLineToEye();
    void calculateProjectedPointOnProjectionPlane();
    void calculateProjectedPointOnScreen();
    void calculateVisibility();
    void setCoordinate(float x, float y, float z);
    void setCoordinate(float point[]);
    void getCoordinate(float arrayToReturnCoordinate[]);
    void printCoordinate();
    void getProjectedVertexOnScreen(int arrayToReturnProjectedVertex[]);
    void roundCoordinates();
    void setColor(char colorToSet[3]); //Set RGB value of the point
    void setColor(char colorToSet1, char colorToSet2, char colorToSet3); //Set RGB value of the point
    bool isEqualTo(RealWorldObjectPoint point); // Check whether realWorldCoordinates are same with that of point
    unsigned char color[3]; // color of the point
    unsigned char seenColor[3]; // color seen after various effects like lightning
    float calculateAndGetDistanceToProjectionPlane(); //calculate distanceToProjectionPlane
    signed char sideOfPointFromProjectionPlane;//is -1 or 1 as of which side the point is of the projection plane. is the sign of value
                                //of equation of projection plane when point is substituted in the x, y and z of equation
    void calculateSideOfPointFromProjectionPlane();
    float calculateAndGetPerpendicularDistanceToProjectionPlane();



  private:
    int projectedVertexOnScreen[3]; //Screen is the plane z = 0
    float realWorldCoordinate[3];
    float projectedVertexOnProjectionPlane[3];
    int directionRatiosForLineToEye[3]; //l, m and n
    float constantRatioForLineToEyeOnProjectionPlane; //'r' on the equation: (x - x1)/l = (y - y1)/m = (z - z1)/n = r (say)
    bool isVisible;
    float distanceToProjectionPlane; //distance between point and projection point
    float perpendicularDistanceToProjectionPlane;

};
```

This header contains the class RealWorldObjectPoint. Any object of this class represent a co-ordinate in 3D space (given by realWorldCoordinate). The projected point of this 3D co-ordinates is calculated from the member function of this class: **calculateProjectedPointOnScreen()** and stored in **projectedVertexOnScreen**. Parameters of line between eye and the point are also calculated in this member function. The color of the point is also stored.

edge.h

```
class Edge {
  friend class Surface;

  public:
    Edge();
    Edge(float a,float b,float c,float d,float e,float f); //Initialize 2 vertex as (a,b,c) , (d,e,f)
    Edge(RealWorldObjectPoint vertex1, RealWorldObjectPoint vertex2); //Initialise 2 vertex as vertex1 and vertex2
    void setCoords(float a,float b,float c,float d,float e,float f); // Set 2 vertex as (a,b,c), (d,e,f)
    void setCoords(float a[], float b[]); //vertex1 = a, vertex2 = b
    void setVertex(RealWorldObjectPoint vertex1, RealWorldObjectPoint vertex2); // Set 2 vertices of the edge
```

```
    void printCoords(); //Display 2 vertices on console
    void calculateParameters(); // Calculate direction ratio, increase in direction, increment, constant for edge on unit increment
                        // and first point on the edge
    void calculateNextPoint(); //Calculate next point on the edge
    void calculateAllPoints(); //Calculate a points on edge
    bool isAVertex(); // Check if the edge is just a single vertex. ie given 2 vertex are same
    vector<RealWorldObjectPoint> allPoints; //all points on the edge


  private:
    RealWorldObjectPoint vertex[2]; //2 end vertices of an edge in clockwise format
    float directionRatio[3];
    int increaseIn; // increase in x, y or z to find next point on the edge. 0, 1 or 2 to increase in x, y abd z respectively
    int increment; //1 or -1 to be added to current points increase in index
    float constantForEdgeOnUnitIncrement; // x-x1/l = r(say) when x - x1 = 1
    float nextPoint[3];
    bool endOfEdge; //true if the point calculated is the last point on the edge


};
```

This header contains the class edge. Edge is the line joining any two 3D coordinates (RealWorldObjectPoint). Direction ratio and other parameters of this line are calculated and stored in this class. All the 3D points in the line are also calculated and stored.

surface.h

```
public:
  Surface(); //Default Constructor
  void addVertex(RealWorldObjectPoint inputVertex); //Add a vertex to the surface
  void setSurfaceFromItsVertices(); //Creates edges from the vertices.
                        //Must be used after all the vertices have been inputted to the surface.
                        //
  void addEdge(Edge e); //Add a edge in the boundary of surface
  void addInnerEdge(Edge e); // Add edge within the region
  void calculateAllEdge(); //Set all Edges in the surface (and points in the edge)
  vector<Edge> allEdge; //all edges in the surface
  void draw(); //Draw Surface on screen after projection
  void calculateSurfaceParameters();// calculate surface normals and  surfaceExtremes
  void setSurfacePropertyMaterialLighting(float property[2][3]); // set surfacePropertyMaterialLighting by passing an array
  void getSurfacePropertyMaterialLighting(float property[2][3]); //returns surfacePropertyMaterialLighting in the array property
  void getSurfaceParameters(float parameters[4]); // returns surfaceParameters in the array parameters passed to the function
  void loadBMPToSurface(const char* fileName); // Load BMP image to the surface

  void printVertexCoordinates(); //prints Coordinates of Vertices of Surface on Screen

  float surfaceExtremes[3][2]; //gives minimum and maximum values of x, y and z coordinates of the surface
                        //1st dimension (i.e. [3]) for x, y and z in order
                        //2nd dimension (i.e. [2]) for minimum and maximum in order
```

```
  private:
    vector<RealWorldObjectPoint> vertex; // vertices in clockwise format
    vector<Edge> edge; // edges in clockwise format
    int beginEdgeId, endEdgeId; // edges between which a edge is chosen for calculating points in the surface
    Edge nextEdge; //next Edge on which points are to be calculated
    float surfaceParameters[4]; //a, b c and d on the equation ax+by+cz = d
    float surfacePropertyMaterialLighting[2][3];

};
```

A Surface is a plane in 3D. It can be defined either through its edges or vertices.For the purpose, there are 2 functions in the class. **addEdge()** and **addVertex()**.These functions add edge and vertex to the surface respectively.The surface parameters (a,b,c,d in the equation ax+by+cz+d=0) are also calculated and stored in the members of this class. All the edges of the surface are also calculated and stored.

cylinder.h

```
class Cylinder{
  public:
    void setTopCentre();
    void setTopCentre(float, float, float);
    void setBottomCentre();
    void setBottomCentre(float,float,float);
    void setTopRadius();
    void setTopRadius(float);
    void setBottomRadius();
    void setBottomRadius(float);
    void setDimension();
    void setDimension(float, float);
    void setStepAngle();
    void setStepAngle(float);
    void setNoOfCylinders();
    void setNoOfCylinders(int);
    void calculateCylinderSurface();

  private:
    int topCentre[3],bottomCentre[3]; //centre coordinates of cylinder - x, y and z
    int dimension[3]; // length, height and depth
    int radius [2];    //radius of cylinder radius[0] for top face and radius [1] for bottom face of cylinder
    float stepAngle;
    int noOfCylinders;
    //RealWorldObjectPoint vertex[73];
    vector<RealWorldObjectPoint> allSurfacePoints;
    vector<Surface> cylinderSurface;



};
```

This header contains the Cylinder class. The member function of this class is used in order to generate the surface of vertical cylinder **calculateCylinderSurface()**. And the remaining functions are used to set the required parameters to generate its surfaces like **setTopCentre()** for setting the top centre of the circular face of the cylinder and so on.

horizontalCylinder.h

```
class HorizontalCylinder{
   public:
      void setFrontFaceCentre();
      void setFrontFaceCentre(float, float, float);
      void setBackFaceCentre();
      void setBackFaceCentre(float,float,float);
      void setFrontRadius();
      void setFrontRadius(float);
      void setBackRadius();
      void setBackRadius(float);
      void setDimension();
      void setDimension(float, float);
      void setStepAngle();
      void setStepAngle(float);
      void setNoOfCylinders();
      void setNoOfCylinders(int);
      void calculateHorizontalCylinderSurface();

   private:
      int frontFaceCentre[3],backFaceCentre[3]; //centre coordinates of cylinder - x, y and z
      int dimension[2]; // length and depth
      int radius [2];    //radius of cylinder radius[0] for front and radius [1] for back
      float stepAngle;
      int noOfCylinders;
      RealWorldObjectPoint frontFaceCentrePoint,backFaceCentrePoint;
      vector<RealWorldObjectPoint> allSurfacePoints;
      vector<Surface> cylinderSurface;



};
```

This header contains the HorizontalCylinder class. The member function of this class is used in order to generate the surface of horizontal cylinder **calculateHorizontalCylinderSurface().** And the remaining functions are used to set the required parameters to generate its surfaces like **setFrontFaceCentre()** for setting the front centre of the circular face of the horizontal cylinder and so on.

cone.h

```
class Cone{
public:
      void setTopCentre();
      void setTopCentre(float,float,float);
      void setBottomCentre();
      void setBottomCentre(float,float,float);
      void setDimension();
      void setDimension(int,int);
      void setStepAngle();
      void setStepAngle(int);
      void calulateConeSurface();

private:
   float topCentre[3],bottomCentre[3];
   float dimension[2];
   float stepAngle;
   RealWorldObjectPoint bottomCentrePoint, topCentrePoint;
   RealWorldObjectPoint vertices[36];
   vector<RealWorldObjectPoint> allConeSurfacePoint;
   vector<Surface> coneSurface;

};
```

This header contains the Cone class. The member function of this class is used in order to generate the surface of cone calulateConeSurface (). And the remaining member functions are used to set the required parameters to generate its surfaces like setDimension () for setting the radius and height of the cone.

imageloader.h

It is downloaded header file. It contains class Image. The class has members to give array of RGB values of each pixels of an image.

lighting.h

```
class Light{
   public:
   Light();
   Light(float pos[3]);//constructor with source position parameters
   Light(float x,float y,float z);

   void setPropertyLight(float amb_r,float amb_g,float amb_b,float diff_r,float diff_g,float diff_b,float
spec_r,float spec_g,float spec_b);
   void getSourcePosition(float arrayToReturnSourcePosition[3]);
   void setSourcePosition(float pos[3]);
   void setPropertyLight(float property[3][3]);
   void calculateSeenColor(Surface& s);
   void setAttenuationConstant(float x,float y,float z);
```

```
    void setAttenuationConstant(float attenuation[3]);
    void calculateAmbientEffect(Surface& s);


    private:
    float attenuationConstant[3];
    float sourcePosition[3];
    float propertyLight[3][3]; // row1 for Ambient RGB, row2 for Diffuse RGB, row3 for Specular
    float attenuation;
    float exponentSpecular;
};
```

This header contains class lighting. This class contains private variables sourcePosition, propertyLight, attenuationConstant, attenuation, exponentSpecular to represent a point light. These properties are set using the fuction setSourcePosition(), setPropertyLight(), setAttenuationConstant(). Then, the function calculateSeenColor() calculates the final R,G,B value of the particular point on the surface. calculateSeenColor() calculates the angle between the surface normal and light for the diffuse part and also calculates the angle between the surface normal and the halfway vector.

collision.h

```
void checkCollision();
```

Under collision header, it has only one member function **checkCollision()** which is used to detect if there is collision in between eye and the surfaces of the models.

screenpixel.h

```
void calculatePointToPlot(); //calculate points which are to be plotted
void plotPixels(); // plot pixels on screen

class ScreenPixel {
    friend void plotPixels();
    friend void calculatePointToPlot();

    public:
        ScreenPixel(); // Default Constructor
      void setColor(unsigned char colorRed, unsigned char colorGreen, unsigned char colorBlue); //set color
      void setColor(unsigned char colorToSet[3]); //set RGB values of color
      void getColor(unsigned char colorToGet[3]); //return color in the array colorToGet passed to the function
      bool isFilled; // true if a vertex is projected to the pixel. False otherwise


    private:
      unsigned char color[3]; //color to be plotted on the screen
      RealWorldObjectPoint pointToPlot; //point whose projection is on the pixel
      float distanceOfPoint;

};
```

This header contains the class ScreenPixels. An object of this class represents a pixel on the screen. This class contains color to be filled onto the pixel, point whose projection is to be displayed on the pixel and distance between the projected point and 3D point. It also contains a Boolean variable to check whether the pixel has been filled or not. Function to plot the pixels has also been included in this header file.

display.h

```
extern bool* specialKeyStates;//yo multiple key press handle garnalai rakheko
extern bool* keyStates;


void handleKeypress(unsigned char key,int x,int y);
void specialKeyOperations();
void keySpecial(int key, int x,int y);
void keySpecialUp(int key,int x, int y);
void keyPressed(unsigned char key,int x, int y);
void keyReleased(unsigned char key, int x,int y);
void keyboardOperation();
void initRendering();
void drawScene();
void handleResize(int w, int h);

void animate();
```

This header contains openGL and Glut functions for displaying and keyboard manipulation.

OUTPUT

LIMITATIONS

- Since the computation is done for each and every pixel/points, the rendering process was slow.
- The object is filled with the scan fill algorithm which is a little buggy and doesn't fill in some places.
- Since, Point collision (Eye) is used , collision isn't detected properly.
- Emissive property of the objects is not modeled.
- Few 3D models are rendered.
- Only opaque objects are modeled.
- No use of mouse for changing direction (Look At).
- Only point light has been used.

FUTURE IMPROVEMENTS

The following improvements could be done to our project in the near future:

- ❖ Multi Threading can be done to increase the computational speed.
- ❖ Transparent objects can be modeled.
- ❖ Filling can be improved.
- ❖ Spot Lights and Cylindrical lights can be used.
- ❖ Mouse can be used for changing direction(Look At).
- ❖ Splines and Curves can be used to model 3D – Objects.

RESOURCES

1) **CodeBlocks 10.05** : The IDE selected for the project development that we have used.
2) **OpenGL graphics library** : The library for drawing the pixel.
3) **OpenGL glut library:** The library for the keyboard operations.

CONCLUSION

By the successful completion of this project, we got very much knowledge about the computer graphics. During the project we learned how to draw a basic 3D models, how can we give the real 3D experience of an object in a 2D screen by applying the lighting effect and rendering effect.

Hence we came to conclude that, we can now work with basic computer graphics programming and design 3D modeling of any real object giving them the virtual realism.

REFERENCES

- Baker, H. &. *Computer Graphics-C Version* (second ed.). Pearson Education.
- Shrestha, S. (2011). *A Text Book Of Engineering Mathematics.* Bhotahity,kathmandu: Vidhyarthi Pustak Bhandar.
- **Nptel Video Lectures** On Computer Graphics.