- **void glLoadIdentity()**

  Sets the current transformations matrix to an identity matrix.

- **void glPushMatrix()**

  void glPopMatrix()

  pushes to and pops from the matrix stack corresponding to the current matrix mode.

- **void glBlendFunc(GLenum sfactor, GLenum dfactor)**

Sfactor - Specifies how the red, green, blue, and alpha source blending factors are computed.

Dfactor - Specifies how the red, green, blue, and alpha destination blending factors are computed.

4.2 Pseudocode

```
#include "stdafx.h"
#include<string.h>
#include<stdlib.h>
#include<Windows.h>
#include<GLUT/glut.h>
#include<stdio.h>
#include<iostream>
#include<math.h>
#define SCENE 10
using namespace std;
void *currentfont;
enum{
FRONTPAGE,ENCRYPTION,ENCRYPTIONINFO,WITHOUTENCRYPTION,EXIT };
int width = 650, height = 650;
int lineheight=500;
int linemargin=500;
int currentheight=400;
void *font =GLUT_BITMAP_HELVETICA_12;
void *fonts[] =
{
GLUT_BITMAP_9_BY_15,
GLUT_BITMAP_TIMES_ROMAN_10,
GLUT_BITMAP_TIMES_ROMAN_24,          // Text Styles
GLUT_BITMAP_HELVETICA_12,
```

```
GLUT_BITMAP_HELVETICA_10,
GLUT_BITMAP_HELVETICA_18,
};
void output(int x, int y, char *string,int j)    {
int len, i;
glColor3f(1.0f,1.0f,0.0f);
glRasterPos2f(x, y);
len = (int) strlen(string);
for (i = 0; i < len; i++)
glutBitmapCharacter(fonts[j], string[i]);
}
void delay(void)   {                          // FUNCTIONS. FOR DELAY
int i,j,k;
for(i=0;i<5000;i++)
{
for(j=0;j<10000;j++);
for(k=0;k<15000;k++);
}
}
void delay1(void)  {
int i;
for(i=0;i<10000;i++);
}
void front_page()                             // FIRST SCREEN - FRONT PAGE
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glClearColor(0.4f,0.2f,0.3f,1.0f);
output(220,550,"VIVEKANANDA INSTITUTE OF TECHNOLOGY",2);
output(320,500,"    A       ",2);
output(220,450,"        MINI PROJECT ON       ",2);
output(220,400," ENCRYPTION AND DECRYPTION ",2);
output(130,200,"Guides:",3);
output(175,180,"Name 1",2);
output(175,150,"Name 2",2);
output(450,200,"By:",3);
output(475,180,"ANJITHA R [1VK15CS004]",2);
output(475,150,"ARCHANA B S [1VK15CS005]",2);
output(275,50,"Press S to start",2);
glutPostRedisplay();
glFlush();
glCallList(SCENE);
glPopMatrix();
glutSwapBuffers();
}
void encryptioninfo()  {
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
```

```
glClearColor(1.0f,0.1f,2.1f,1.0f);
output(250,550,"The message is",2);
output(270,500,"    hii ",2);
output(220,450,"        The key is      ",2);
output(220,400,"        p=11,q=13       ",2);
output(220,100,"Press W w to view without using encryption and decryption",2);
output(220,50,"Press E e to view RSA encryption and decryption ",2);
 glFlush();
glCallList(SCENE);
glPopMatrix();
glutSwapBuffers();
}
Glint                                                   /*Movement angle*\
movement_angle=0,packet_angle=0,head_angle=0,packet_angle1=0,head_angle1=0,packe
t_angle2=0,head_angle2=0,packet_angle3=0,head_angle3=0,packet_angle4=0;
GLdouble mov_speed = 1;
GLint
head_angle4=0,packet_angle5=0,head_angle5=0,packet_angle6=0,head_angle6=0,packet_
angle7,head_angle7=0,packet_angle8=40;
GLint                                                   /* Movement angles*\
movement_angle1=0,arrow_angle=0,rev_arrow_angle=0,phy_header_angle=0,rev_phy_he
ader_angle=0,rev_phy_header_angle1=0,phy_header_angle1=0,analog_sig_angle=0,rev_a
nalog_sig_angle = 0;
void animation_encryp(void)  {
if ((movement_angle += mov_speed) >= 600)
movement_angle = 600;
if ((arrow_angle += mov_speed) >= 150)
arrow_angle = 150;
if(arrow_angle==150)
if ((phy_header_angle += mov_speed) >= 100)
phy_header_angle = 100;
if(phy_header_angle==100)
if ((phy_header_angle1 += mov_speed) >= 100)
phy_header_angle1 = 100;
if(phy_header_angle1==100)
if ((analog_sig_angle += mov_speed) >= 100)
analog_sig_angle = 100;
if(analog_sig_angle==100)
if ((movement_angle1 += mov_speed) >= 420)
movement_angle1 = 420;
if(movement_angle1 ==420)
if ((rev_analog_sig_angle += mov_speed) >= 100)
rev_analog_sig_angle = 100;
if(rev_analog_sig_angle ==100)
if ((rev_phy_header_angle += mov_speed) >= 200)
rev_phy_header_angle = 200;
if(rev_phy_header_angle ==200)
if ((rev_phy_header_angle1 += mov_speed) >= 100)
rev_phy_header_angle1 = 100;
if(rev_phy_header_angle ==200)
```

```
if ((rev_arrow_angle += mov_speed) >= 100)
rev_arrow_angle = 100;
glutPostRedisplay();
}
void computer(){
glColor3f(0.75,0.85,0.65);//keyboard
glBegin(GL_QUADS);
glVertex2f(55.0,340.0);
glVertex2f(145.0,340.0);
glVertex2f(150.0,350.0);
glVertex2f(60.0,350.0);
glEnd();
glColor3f(0.75,0.85,0.65);//cabinet
glBegin(GL_LINE_LOOP);
glVertex2f(60.0,355.0);
glVertex2f(150.0,355.0);
glVertex2f(150.0,370.0);
glVertex2f(60.0,370.0);
glEnd();
glColor3f(0.75,0.85,0.65);
glBegin(GL_LINE_LOOP);
glVertex2f(75.0,380.0);
glVertex2f(135,380.0);
glVertex2f(135.0,430.0);
glVertex2f(75.0,430.0);
glEnd();
glColor3f(0.7,0.8,0.6);
glBegin(GL_QUADS);
glVertex2f(80.0,385.0);
glVertex2f(130.0,385.0);
glVertex2f(130.0,425.0);
glVertex2f(80.0,425.0);
glEnd();
glColor3f(0.75,0.85,0.65);
glBegin(GL_LINES);
glVertex2f(90.0,370.0);
glVertex2f(90.0,380.0);
glVertex2f(120.0,370.0);
glVertex2f(120.0,380.0);
glEnd();
}
static long gcd(long m,long n)    {
long r;
        while(n!=0)
        {
                r=m%n;
                m=n;
                n=r;
        }
        return m;
```

```
}
void encrypt(int x,int y,int z){
string msg="hii";
long encrypted[100];
long num[100];
        int p=11,q=13,phi,i,len,e,d,j;
        int n=p*q;
        phi=(p-1)*(q-1);
        for(i=2;i<phi;i++)
                if(gcd(i,phi)==1)
                        break;
        e=i;
        for(i=2;i<phi;i++)
                if((e*i-1)%phi==0)
                        break;
        d=i;
        len=msg.length();

        for(i=0;i<len;i++)
                num[i]=msg[i];

        for(i=0;i<len;i++)
        {
        encrypted[i]=1;
        for(j=0;j<e;j++)
        {
                encrypted[i]=(encrypted[i]*num[i]%n);
        }
        }
        glRasterPos3i(x,y,z);
        for(i=0;encrypted[i]!='\0';i++){
                glutBitmapCharacter(font,encrypted[i]);
        }
        glutPostRedisplay();
        glFlush();
}
void cipher() {
glColor3f(0.0f,0.0f,1.0f);
glPushMatrix();
glScalef(40,20,.5);
glTranslatef(3,14,0);
glutWireCube(2);
encrypt(-1,0,0);
glPopMatrix();
}
void message_data() {
glColor3f(1.0f,1.0f,0.0f);
glPushMatrix();
glScalef(40,20,.5);
glTranslatef(3,14,0);
```

```
glutWireCube(2);
output(-1,0,"hii",2);
glPopMatrix();
}
void message_key(void) {
glColor3f(0.0f,1.0f,0.0f);
glPushMatrix();
glScalef(20,20,.5);
glTranslatef(3,14,0);
glutWireCube(2);
output(-1,0," p=11,q=13",2);
glPopMatrix();
}
void decrypt(float x,float y,float z) {
        string msg="[vv";
long encrypted[100];
long decrypted[100];
long num[100];
        int p=11,q=13,phi=0,i=0,len=0,e=0,d=0,j=0;
        int n=p*q;
        phi=(p-1)*(q-1);
        for(i=2;i<phi;i++)
                if(gcd(i,phi)==1)
                        break;
        e=i;
        for(i=2;i<phi;i++)
                if((e*i-1)%phi==0)
                        break;
        d=i;
        len=msg.length();

        for(i=0;i<len;i++)
                num[i]=msg[i];
        glRasterPos3i(x,y,z);
        for(int i=0;i<len;i++)
        {
                decrypted[i]=1;
       for(int j=0;j<d;j++)
       {
                decrypted[i]=(decrypted[i]*msg[i]%n);

       }
       for(i=0;i<=2;i++){

                glutBitmapCharacter(font,(char)decrypted[i]);
       }
       glutPostRedisplay();
       glFlush();
       }
```

```
void message_data3() {
glColor3f(1.0f,1.0f,0.0f);
glPushMatrix();
glScalef(40,20,.5);
glTranslatef(3,14,0);
glutWireCube(2);
output(-1,0,"ERROR",2);
glPopMatrix();
}
void withoutencryption() {
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glClearColor(0.5f,0.5f,0.5f,1.0f);
glColor3f(0.0f,1.0f,0.0f);
output(120,640,"WITHOUT USING RSA ENCRYPTION AND DECRYPTION",2);
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
glTranslatef(-30,200,0);
output(75,440,"Sender",2);
computer();
glPopMatrix();
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
output(565,630,"Receiver",2);
computer_dest();
glPopMatrix();
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
output(320,190,"Intruder",2);
computer_mid();
glPopMatrix();
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
glScalef(70,40,.5);
glTranslatef(1.5,7,0);
glutWireCube(2);
glPopMatrix();
if(phy_header_angle1!=100) {
glPushMatrix();
glTranslatef(0,-phy_header_angle1,0);
glPushMatrix();
glTranslatef(0,-arrow_angle,0);
glTranslatef(0,150,0);
message_data();
glPopMatrix();
glPushMatrix();
glTranslatef(phy_header_angle,0,0);
glTranslatef(-100,0,0);
glPopMatrix();
glPopMatrix();
```

```
}
glPushMatrix();
glTranslatef(movement_angle1,0,0);
if(phy_header_angle1==100)
{
glPushMatrix();
if(movement_angle1>=420) {
glTranslatef(0,rev_analog_sig_angle,0);
}
glTranslatef(0,-analog_sig_angle,0);
if(rev_analog_sig_angle!=100)
cipher2();
glPopMatrix();
}
glPopMatrix();
glPushMatrix();
glTranslatef(movement_angle1,0,0);
if(phy_header_angle1==100) {
glPushMatrix();
glScalef(50,30,.5);
glTranslatef(2.5,2.5,0);
glutWireCube(2);
glPopMatrix();
}
glPopMatrix();
if(rev_analog_sig_angle==100) {
glPushMatrix();
glTranslatef(450,rev_phy_header_angle1,0);
glPushMatrix();
glTranslatef(0,rev_arrow_angle,0);
glTranslatef(0,0,0);
message_data3();
glPopMatrix();
glPushMatrix();
glTranslatef(rev_phy_header_angle,0,0);
glTranslatef(-10,0,0);
glPopMatrix();
glPopMatrix();
}
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
glTranslatef(450,0,0);
glScalef(70,40,.5);
glTranslatef(1.5,7,0);
glutWireCube(2);    //right layer
glPopMatrix();
animation_encryp();
glFlush();
glutSwapBuffers();
}
```

```
void encryption() {
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glColor3f(1.0f,1.0f,1.0f);
output(150,640,"ENCRYPTION AND DECRYPTION PROCESS",2);
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
glTranslatef(-30,200,0);
output(75,440,"Sender",2);
computer();
glPopMatrix();
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
output(565,630,"Receiver",2);
computer_dest();
glPopMatrix();
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
output(320,190,"intruder",2);
computer_mid();
glPopMatrix();
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
glScalef(70,40,.5);
glTranslatef(1.5,7,0);
glutWireCube(2);
glPopMatrix();
if(phy_header_angle1!=100) {
glPushMatrix();
glTranslatef(0,-phy_header_angle1,0);
glPushMatrix();
glTranslatef(0,-arrow_angle,0);
glTranslatef(0,150,0);
message_data();
glPopMatrix();
glPushMatrix();
glTranslatef(phy_header_angle,0,0);
glTranslatef(-100,0,0);
message_key();
glPopMatrix();
glPopMatrix();
}
glPushMatrix();
glTranslatef(movement_angle1,0,0);
if(phy_header_angle1==100) {
glPushMatrix();
if(movement_angle1>=420) {
glTranslatef(0,rev_analog_sig_angle,0);
}
glTranslatef(0,-analog_sig_angle,0);
```

```
if(rev_analog_sig_angle!=100)
cipher();
glPopMatrix();
}
glPopMatrix();
glPushMatrix();
glTranslatef(movement_angle1,0,0);
if(phy_header_angle1==100) {
glPushMatrix();
glScalef(50,30,.5);
glTranslatef(2.5,2.5,0);
glutWireCube(2);
glPopMatrix();
}
glPopMatrix();
if(rev_analog_sig_angle==100) {
glPushMatrix();
glTranslatef(450,rev_phy_header_angle1,0);
glPushMatrix();
messagedata2();
glPopMatrix();
glPushMatrix();
glTranslatef(rev_phy_header_angle,0,0);
glTranslatef(-10,0,0);
if(rev_phy_header_angle<200)
message_key();
glPopMatrix();
glPopMatrix();
}
glColor3f(1.0f,1.0f,1.0f);
glPushMatrix();
glTranslatef(450,0,0);
glScalef(70,40,.5);
glTranslatef(1.5,7,0);
glutWireCube(2);//right layer
glPopMatrix();
animation_encryp();
glFlush();
glutSwapBuffers();
}
void myinit() {
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,700.0,0.0,700.0);
}
```

```
void display() {
glClearColor(0.0,0.0,0.0,0.0);
glClear(GL_COLOR_BUFFER_BIT);
glClearColor(1.0,0.0,0.4,1.0);
front_page();
glFlush();
glutSwapBuffers();
}
void key (unsigned char key, int x, int y) {
switch(key) {
case 'S' :
case 's' :glutDisplayFunc(encryptioninfo);
          break;
case 'W' :
case 'w' :glutDisplayFunc(withoutencryption);
          break;
case 'E':
case 'e':glutDisplayFunc(encryption);
          break;
case 'q':
case 'Q':
          exit(0);
 }
 glutPostRedisplay();
}
static void menu(int mode) {
switch (mode) {
case FRONTPAGE: glutDisplayFunc(front_page);
          break;
case ENCRYPTIONINFO: glutDisplayFunc(encryptioninfo);
          break;
case WITHOUTENCRYPTION: glutDisplayFunc(withoutencryption);
          break;
case ENCRYPTION: glutDisplayFunc(encryption);
         break;
case EXIT: exit(0);
}
glutPostRedisplay();
}
void main(int argc, char** argv) {
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(1000,700);
glutInitWindowPosition(0,0);
glClearColor(0.9f,0.5f,0.2f,1.0);
glutCreateWindow("ENCRYPTION");
glutKeyboardFunc(key);
myinit();
glutDisplayFunc(display);
glutCreateMenu(menu);
```

```
glutAddMenuEntry("Front Page",.FRONTPAGE);
glutAddMenuEntry("Encryptioninfo",ENCRYPTIONINFO);
glutAddMenuEntry("Without Encryption ",WITHOUTENCRYPTION);
glutAddMenuEntry("Encryption",ENCRYPTION);
glutAddMenuEntry("Exit", EXIT);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
}
```