Geometric Transformations

Submitted as a

Project in
Computer Graphics
for
Semester V

For partial completion of
B-Tech in Computer Science & Engineering

By:
Vaibhav Bajpai (0509713060)
Yadvendra (0509710121)

Under the guidance of

Sir. Anand Srivastava
Department of Computer Science & Engineering
Galgotias College of Engineering Technology
Greater Noida

Overview

A **transformation** in elementary terms is any of a variety of different functions from geometry, such as rotations, reflections and translations. These can be carried out in Euclidean space, particularly in dimensions 2 and 3.

Transformations:

- Geometric Transformations
- Coordinate Transformations

Geometric Transformations:

- Translation
- Rotation
- Scaling
- Reflection
- Shear

Feasibility Study

The project uses an arbitarily drawn triangle as a sample polygon to which successive geometric transformations are applied.

Transformations Supported:

- Translation
- Reflection
- Scaling

Technologies:

- **Language:** C++
  Compiler: GCC(g++)
- **Graphic Library:** SDL(Simple Direct Media Layer)
- **Supported O/S:** Linux, Windows, OSX

Key Features

- **32-bit C++ Compiler**

  The project uses GCC instead of 16-bit Turbo C++ compiler extending the address space of the project beyond 16KiB to support high resolution as 800x600 to display polygons and preventing stack overflows in case of flood-fill subroutines that extensively use recursion to fill the areas.

- **Cross-Platform Graphic Library**

  The project uses **SDL(Simple Direct Media Layer)** instead of **graphics.h** to implement low level pixel operations to allow the project to be easily ported across different platforms

Simple Direct Media Layer

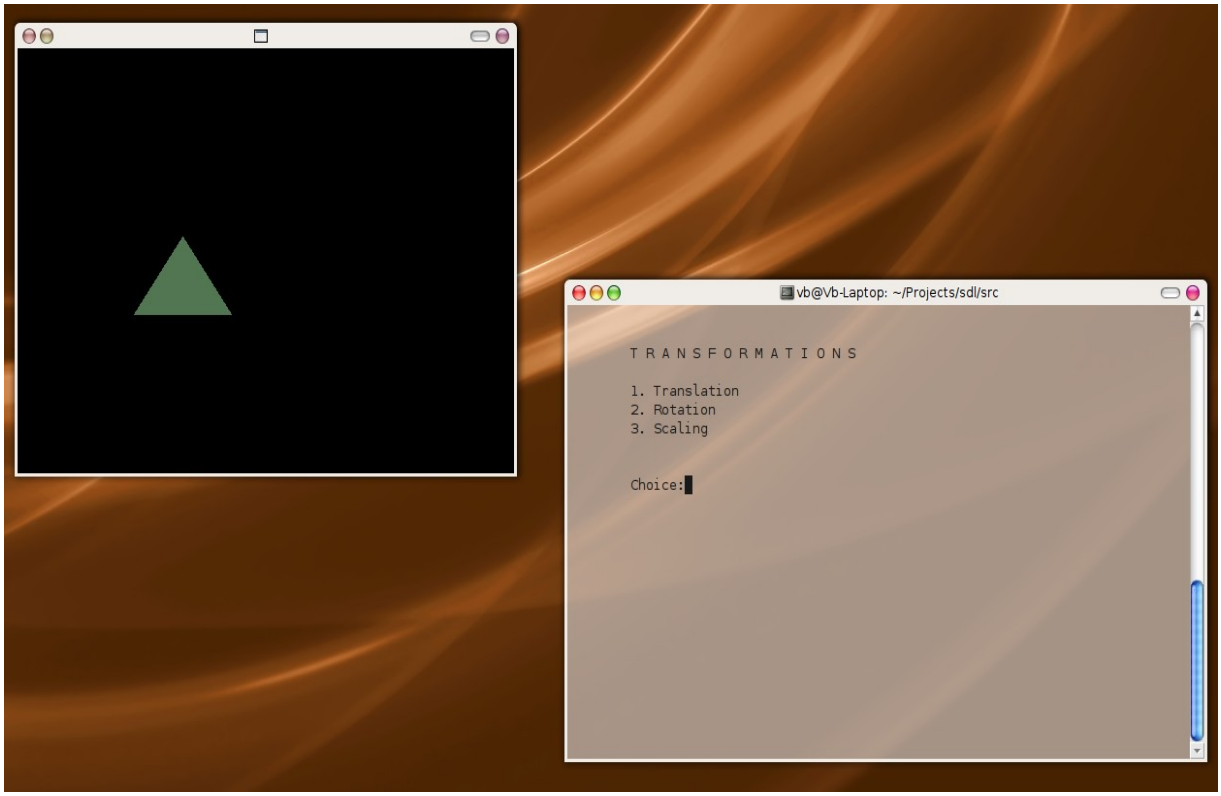
- Simple DirectMedia Layer is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video frame buffer.

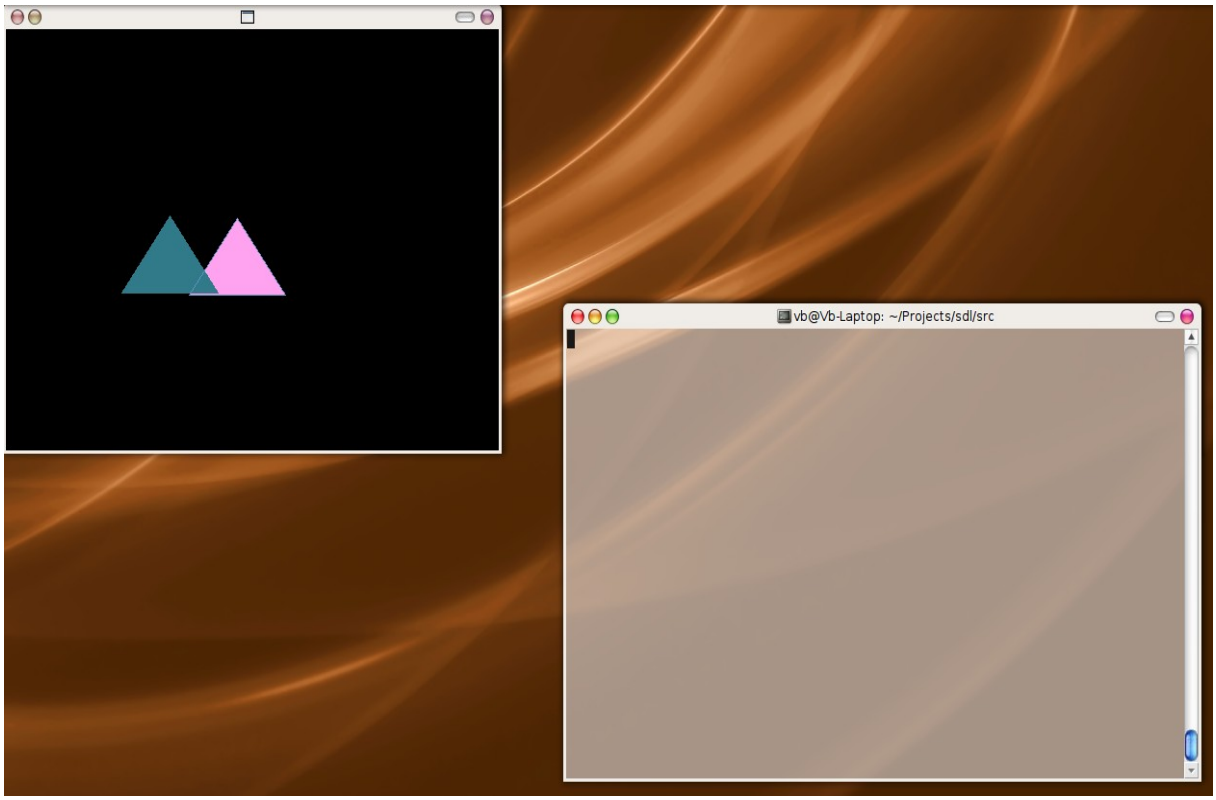
- SDL supports Linux, Windows, Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, and QNX

- SDL is written in C, but works with C++ natively . SDL is distributed under GNU LGPL version 2.

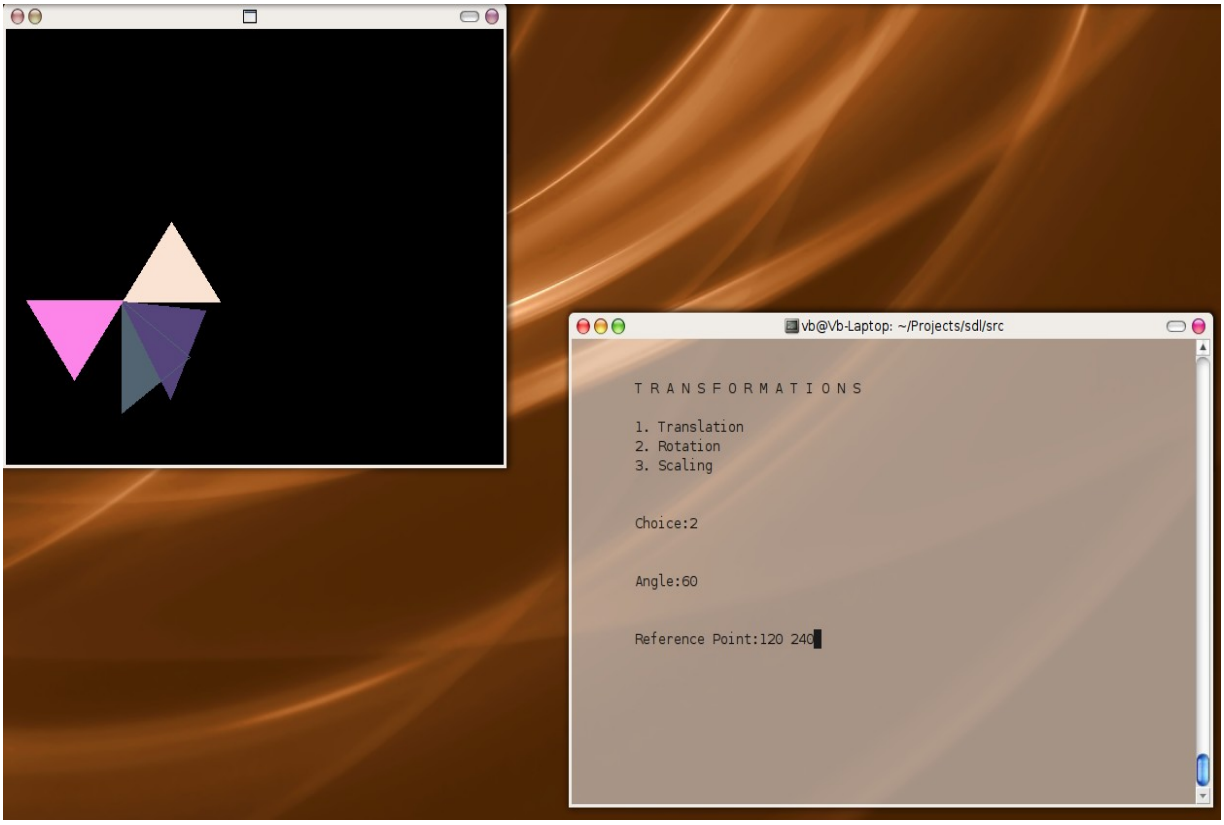
.

Screenshots





Screenshots



Source Code

```
#include "SDL/SDL.h"
#include <iostream>
#include <cmath>
#include <ctime>
#define round(x) (int)((x)+0.5)
#define PI 3.14159265
using namespace std;
SDL_Surface* pSurface;

typedef float Matrix3x3[3][3];
Matrix3x3 Matrix;

enum
{
  SCREENWIDTH = 512,
  SCREENHEIGHT = 384,
  SCREENBPP = 0,
  SCREENFLAGS = SDL_ANYFORMAT
} ;

struct Point
{
     int xco;
         int yco;
};

class Transformation
{

  public:

  static void Init()
  {
    SetIdentity(Matrix);
  }

  static void SetIdentity(Matrix3x3 m)
  {
    for(int i=0;i<3;i++)
            for(int j=0;j<3;j++)
                    m[i][j] = (i==j);
  }
```

```
static void PreMultiply(Matrix3x3 a, Matrix3x3 b)
{
  Matrix3x3 tmp;

  for(int r=0;r<3;r++)
    for(int c=0;c<3;c++)
      tmp[r][c] = a[r][0]*b[0][c] + a[r][1] *b[1][c] + a[r][2] *b[2][c];

  for(int r=0; r<3; r++)
    for(int c=0;c<3;c++)
            b[r][c] = tmp[r][c];
}

static void Translate(int tx, int ty)
{
  Matrix3x3 m;

  SetIdentity(m);

  m[0][2] = tx;
  m[1][2] = ty;

  PreMultiply(m,Matrix);
}

static void Rotate(float a, Point p)
{
  Matrix3x3 m;

  SetIdentity(m);

  a = a * PI / 180;

  m[0][0] = cosf(a);
  m[0][1] = -sinf(a);
  m[0][2] = p.xco * (1- cosf(a)) + p.yco * sinf(a);
  m[1][0] = sinf(a);
  m[1][1] = cosf(a);
  m[1][2] = p.yco * (1 - cosf(a)) - p.xco * sinf(a);

  PreMultiply(m,Matrix);
}

static void Scale(float sx, float sy, Point p)
{
  Matrix3x3 m;

  SetIdentity(m);

  m[0][0] = sx;
  m[0][2] = (1-sx) * p.xco;
  m[1][1] = sy;
  m[1][2] = (1-sy) * p.yco;

  PreMultiply(m,Matrix);
}
```

```
  static void TransformPoints(const int num, Point *P)
  {
    float tmp;

    for(int k = 0; k<num; k++)
    {
      tmp = Matrix[0][0] * P[k].xco + Matrix[0][1] * P[k].yco + Matrix[0][2];

      P[k].yco = Matrix[1][0]*P[k].xco + Matrix[1][1]*P[k].yco + Matrix[1][2];

      P[k].xco = (int)tmp;
    }
  }
};

class Start
{
  public:

  static const int num=4;

  static Point P[4];


  static void Menu()
  {

      system("clear");

      SDL_Flip(pSurface);

      cout<<"\n\n\tT R A N S F O R M A T I O N S \n\n";

      int ch;

      Transformation::Init();

      cout << "\t1. Translation\n";
      cout << "\t2. Rotation\n";
      cout << "\t3. Scaling\n";

      cout << "\n\n\tChoice:";

      cin>>ch;


      switch(ch)
      {
            case 1:
                int tx,ty;

                cout<<"\n\n\tTranslation Distances:";
                cin>>tx>>ty;

                Transformation::Translate(tx,ty);

                break;
```

```
            case 2:
                 Point p1;
                 float angle;

                 cout<<"\n\n\tAngle:";
                 cin>>angle;

                 cout<<"\n\n\tReference Point:";
                 cin>>p1.xco>>p1.yco;

                 Transformation::Rotate(angle, p1);

                 break;

             case 3:
                 float sx,sy;
                 Point p2;

                 cout<<"\n\n\tScaling Factors:";
                 cin>>sx>>sy;

                 cout<<"\n\n\tReference Point:";
                 cin>>p2.xco>>p2.yco;

                 Transformation::Scale(sx,sy,p2);

        }

}

static void Draw()
{
   srand((unsigned int)time(NULL));

   SDL_Color color ;
   color.r = rand ( ) % 256 ;
   color.g = rand ( ) % 256 ;
   color.b = rand ( ) % 256 ;

   PolygonDraw(num,P,color);

   int xc = (P[0].xco + P[1].xco + P[2].xco)/3;
   int yc = (P[0].yco + P[1].yco + P[2].yco)/3;

   PolygonFill(xc,yc,color);
}

static void PolygonDraw(int n, Point points[],SDL_Color color)
{
  if(n>=2)
  {
     for(int count=0;count<n-1;count++)
          lineDDA(&points[count],&points[count+1],color);
  }
}
```

```
  static void PolygonFill(int xc,int yc, SDL_Color color)
  {
        SDL_Color color1 ;              // Black
        color1.r = 0 ;
        color1.g = 0 ;
        color1.b = 0 ;

        FloodFill(xc,yc,color,color1);
  }

static void lineDDA( struct Point * p1, struct Point *p2, SDL_Color color)
{
  int dx,dy,steps,k;
  float xinc,yinc,x,y;

  dx=p2->xco - p1->xco;
  dy=p2->yco - p1->yco;

  if(abs(dx) > abs(dy))
        steps=abs(dx);
  else
        steps=abs(dy);

  xinc=(float)dx/(float)steps;
  yinc=(float)dy/(float)steps;
  x=p1->xco;
  y=p1->yco;

  SetPixel(pSurface,round(x),round(y),color);

  for(k=0;k<steps;k++)
  {
        x+=xinc;
        y+=yinc;
        SetPixel(pSurface,round(x),round(y),color);
  }
}

static void FloodFill(int x, int y, SDL_Color fill, SDL_Color oldcolor)
{
  SDL_Color current = GetPixel(pSurface,x,y);

  if(CompareColor(current,oldcolor))
  {
          SetPixel(pSurface,x,y,fill);
          FloodFill(x+1,y,fill,oldcolor);
          FloodFill(x-1,y,fill,oldcolor);
          FloodFill(x,y+1,fill,oldcolor);
          FloodFill(x,y-1,fill,oldcolor);
  }
}
static bool CompareColor(SDL_Color c1, SDL_Color c2)
{
  if(c1.r == c2.r && c1.g == c2.g && c1.b == c2.b)
          return true;
  else
          return false;
}
```

```
static void SetPixel ( SDL_Surface* pSurface , int x , int y , SDL_Color color )
{
    //convert color
    Uint32 col = SDL_MapRGB ( pSurface->format , color.r , color.g , color.b ) ;

    //determine position
    char* pPosition = ( char* ) pSurface->pixels ;

    //offset by y
    pPosition += ( pSurface->pitch * y ) ;

    //offset by x
    pPosition += ( pSurface->format->BytesPerPixel * x ) ;

    //copy pixel data
    memcpy ( pPosition , &col , pSurface->format->BytesPerPixel ) ;
}

static SDL_Color GetPixel ( SDL_Surface* pSurface , int x , int y )
{
    SDL_Color color ;
    Uint32 col = 0 ;

    //determine position
    char* pPosition = ( char* ) pSurface->pixels ;

    //offset by y
    pPosition += ( pSurface->pitch * y ) ;

    //offset by x
    pPosition += ( pSurface->format->BytesPerPixel * x ) ;

    //copy pixel data
    memcpy ( &col , pPosition , pSurface->format->BytesPerPixel ) ;

    //convert color
    SDL_GetRGB ( col , pSurface->format , &color.r , &color.g , &color.b ) ;
    return ( color ) ;
}

};

class SDLInit
{
  public:

  static void Init()
  {
      //initialize systems
      SDL_Init ( SDL_INIT_VIDEO ) ;

      //set our at exit function
      atexit ( SDL_Quit ) ;
```

```
        //create a window
        pSurface = SDL_SetVideoMode ( SCREENWIDTH , SCREENHEIGHT ,

                                      SCREENBPP , SCREENFLAGS ) ;

        //declare event variable
        SDL_Event event ;

        //message pump
        for ( ; ; )
        {
              //look for an event
              if ( SDL_PollEvent ( &event ) )
              {
                //an event was found
                if ( event.type == SDL_QUIT ) break ;
              }

              Start::Draw();

              Start::Menu();

              Transformation::TransformPoints(Start::num,Start::P);

              Start::Draw();

        }//end of message pump

        //done
    }


};

Point Start::P[4] = {{120,240}, {220,240} , {170,170}, {120,240}};

int main()
{
   SDLInit::Init();
   return ( 0 ) ;
}
```