

שם המגיש : דור סרוסי

תעודת זהות : 205404528

כיתה : י''ב גמיש 3

בית ספר : אורט עפולה

שם המנחה : ליאור גורודיסקי

תאריך הגשה : 11.5.2012

בפרויקט שבניתי בחרתי לבנות את משחק הפגיעה במטרה בסגנון של פגיעה בצלחת.

המשחק בנוי בעיקרו משלושה גופים , הרובה , הצלחת והכדור.

המצב ההתחלתי של המשחק: הצלחת נמצאת בצידו השמאלי של המסך (במקום בו אנו לא יכולים לראות אותה) , בלחיצה על כפתור הEnter הצלחת מתחילה את תנועתה על ציר הX במהירות מסוימת. בלחיצה על כפתור הרווח Space , נורה מהרובה כדור שעף לכיוון הכוונת.

מטרת המשחק היא לפגוע ולהרוס כמה שיותר צלחות בכמות הכדורים הנתונה.

בכל פגיעה בצלחת המשתמש צובר נקודה . המשתמש יכול לצבור עד ל-10 נקודות (כמספר הכדורים הנתונים לו ברובה).

לשחקן יש את כל הזמן שהוא יצטרך עד לסיום מחסנית הרובה.

ברגע שהמחסנית מתרוקנת , המשחק נגמר , ולשחקן מוצגת כמות הנקודות שצבר.

המשחק עדיין לא גמור לכן, לשחקן תהיה אפשרות לאחר שסיים את המשחק לשחק שוב מבלי לצאת ולהיכנס למשחק. התחלת המשחק מחדש מאפסת את הנקודות ומחזירה את המחסנית למצבה ההתחלתי (10 כדורים).

במסך הפתיחה שיהיה יופיעו 3 אפשרויות :

1.Play- להתחיל במשחק.

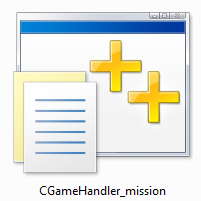
2. How to play- הוראות המשחק.

3. Exit- יציאה מהמשחק.

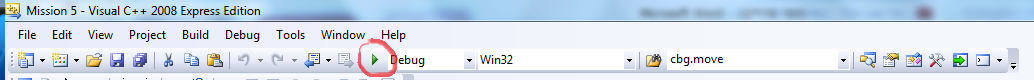
המשחק מבוסס על משחקי הפגיעה במטרה שבהם על השחקן לפגוע במטרה(לעיתים נעה ולעיתים לא) באמצעות כלים שונים . במשחקים אלו יכולים לשחק שחקן ויותר.



1. היכנסו לתיקיית הHitThePlate :



2. הפעל את הקובץ שנקרא CGameHandler\_mission :

3. לאחר הפעלת הקובץ CGameHandler\_mission תפתח התכונה Microsoft visual C++ 2008, כדי להריץ את המשחק יש ללחוץ על החץ הירוק שנמצא בשורת המשימות בחלק העליון של התוכנה

4.זהו שלב עתידי שעדיין איננו איך יהיה במשחק הגמור :

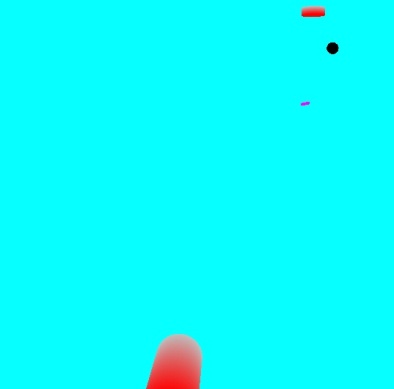
כעת יפתח המשחק ובמסך הראשי מוצגות לפנייך 3 אופציות.

1.Play- להתחיל במשחק.

2. How to play- הוראות המשחק.

3. Exit- יציאה מהמשחק.

5.לאחר לחיצה על המקש 1 תגיע למצב ההתחלתי וכעת עלייך ללחוץ על הEnter והמשחק יתחיל.

בהצלחה ! ☺

הפרויקט נבנה בשפת C++ בסביבת העבודה Microsoft visual C++ 2008 ביחד עם חבילה גרפיתOpenGL (Glut) אשר ביחד נתנו את האפשרות להציג בצורה ויזואלית על המסך את האובייקטים (הרובה, הכדור, הצלחת) אשר יצרו בסוף את התוצר הסופי, המשחק.

**לבניית הפרויקט נעזרתי ב4 מחלקות בסיס:**

CPoint – מחלקת נקודה , שמייצרת נקודות במסך.  
CPolygon – מחלקת פוליגון, המחלקה מחברת בין מספר נקודות ויוצרת בכך את הפוליגון.  
CModel – מחלקת מודל , שמחברת מספר פוליגונים יחד ליצירת אובייקט תלת מימדי.

**קבצי הפרוייקט:**

Ccylinder.h

Ccylinder.cpp

תפקיד מחלקה זו לממש גליל תלת מימדי . הגליל יבוא לידי ביטוי כרובה , כצלחת וככדור.

**מסך פתיחה :**

עדיין לא בנוי , המסך יהיה במשחק הגמור והוא עתידי להיבנות.

**הכוונת:**

CCircle.h

CCircle.cpp

**ניהול המשחק(הדפסת אובייקטים, חלק לוגי):**

CGameHandler.h

CGameHandler.cpp

**יצירת אובייקט תלת-מימד:**

CModel.h

Cmodel.cpp

CMyOpenGLInit.h

**יצירת נקודה:**

CPoint.h

CPoint.cpp

**יצירת מישור:**

CPolygon.h

CPolygon.cpp

**העבודה על הפרויקט הייתה מהנה מאוד , העבודה הייתה מאתגרת , כיפית , לעיתים קלה ולעיתים קשה. למדתי המון משפת הC++ (עד כה ידעתי את שפת הC# רק) , למדתי על השימוש בצורות תלת ממדיות , ודו ממדיות.**

**העבודה הייתה כיפית ומהנה בעיקר מהבחינה שאפשר לבדוק אם מה שעשית נכון ע"י הרצה פשוטה. אם ההרצה הייתה מוצלחת , הדברים היו מוצגים לי על המסך (הצורות והכותרות).**

**אני מרוצה בכך שעמדתי בזמנים שהיו נתונים לי , זהו פרויקט ראשון שלי בסדר גודל כזה בתחום המחשבים ואני מרוצה ממנו מאוד.**

**הפרויקט עדיין לא גמור , נשאר לי לסיים כמה חלקים כמו , יציאת הצלחות בסדר מסויים , שינוי תנועתן וצבירת הנקודות. חלקים אלו יהיו מוכנים בזמן לבחינת הבגרות ולמועד שבו אני אצטרך להגיש את הפרויקט.**

**הפרויקט הוציא ממני צד יצירתי וחשיבתי , הוא גרם לי לחשוב בצורות יותר יצירתיות ומאתגרות ולראות את עולם העיצוב בעין שונה , אם זה לראות משחק באינטרנט ולחשוב על איך בנו אותו , ועד לנסות להבין כמה קשה היה לבנות את הפוליגונים שמוצגים במשחק כלשהו.**

**הקודים עדיין לא גמורים והם עתידיים להשתנות עד מועד ההגשה האחרון.**

**CCircle.h :**

#pragma once

#include "cpolygon.h"

class CCircle :

public CPolygon

{

public:

CCircle(void);

CCircle(int number\_of\_points);

CCircle(int number\_of\_points, double radius);

void CreateCircle(int number\_of\_points, double radius);

};

**CCircle.cpp :**

#include "CCircle.h"

#include <math.h>

CCircle::CCircle(void)

{

CreateCircle(30,0.30);

}

CCircle::CCircle(int number\_of\_points)

{

}

CCircle::CCircle(int number\_of\_points, double radius)

{

}

void CCircle::CreateCircle(int number\_of\_points, double radius)

{

SetNumberOfPoints(number\_of\_points);

double alpha = 2\*3.14/number\_of\_points;

for(int i=0; i<number\_of\_points; i++)

{

double x\_i = radius\*cos(alpha\*i);

double y\_i = radius\*sin(alpha\*i);

double z\_i = 0;

SetOnePointCoordinates(i,x\_i,y\_i,z\_i);

}

SetPolygonColor(80/128.0,80/128.0,80/128.0);

}

**CCylinder.h :**

#pragma once

#include "cmodel.h"

class CCylinder :

public CModel

{

public:

CCylinder(void);

};

**CCyilnder.cpp :**

#include "CCylinder.h"

#include "CCircle.h"

// cylinder is with circle in xy with radius=1 and depth = 1 (between 0 to 1)

CCylinder::CCylinder(void)

{

CCircle circle, circle2;

circle.MovePolygonCenter(0,0,1);

circle2.MovePolygonCenter(0, 0, 0);

circle.SetPolygonColor(60,0,0);

CPolygon part\_of\_cylinder(4);

SetNumberOfPolygons(circle.GetNumberOfPoints()+2);

for (int point\_index=0; point\_index<circle.GetNumberOfPoints(); point\_index++)

{

int next\_point\_index = point\_index+1;

if(next\_point\_index==30)

{

next\_point\_index=0;

}

part\_of\_cylinder.SetOnePoint(0,circle.GetPoint(next\_point\_index));

part\_of\_cylinder.SetOnePoint(1,circle.GetPoint(point\_index));

part\_of\_cylinder.SetOnePoint(2,circle2.GetPoint(point\_index));

part\_of\_cylinder.SetOnePoint(3,circle2.GetPoint(next\_point\_index));

SetOnePolygon(point\_index, part\_of\_cylinder);

}

SetOnePolygon(circle.GetNumberOfPoints(), circle);

SetOnePolygon(circle.GetNumberOfPoints()+1, circle2);

}

**CGameHandler.h :**

#pragma once

class CGameHandler

{

public:

//the beginning

CGameHandler(){};

void StartOfGame();

//on each frame:

void DrawScene();

void LogicPart();

void WriteText();

void KeyPressed(char key);

void SpecialKeyPressed(char key);

//getting backgroundColor

float GetBeckgroundColorR();

float GetBeckgroundColorG();

float GetBeckgroundColorB();

private:

//other procedures

void ChangeBackgroundColors(float R, float G, float B);

//internal variables:

float BGColor\_R,BGColor\_G,BGColor\_B; //BackGround colors.

int font,def\_font; //determine the font to the text written.

char\* WrongKeyText;

};

**CGameHandler.cpp :**

#include <math.h>

#include <stdio.h>

#include "glut.h"

#include "CGameHandler.h"

#include "CModel.h"

#include "CCube.h"

#include "CPolygon.h"

#include "CCircle.h"

#include "CSquare.h"

#include "CCylinder.h"

CCube cube;

bool show;

double rotate\_speed;

bool stop;

double move;

bool stopmove;

double dor;

int frame\_number;

CCircle aim , opening;

CPolygon part\_of\_cylinder;

CCylinder cylinder , plate;

int point\_index;

int bullets\_number;

const int num\_of\_bullets = 10;

CCylinder bullets[num\_of\_bullets];

CPoint bullet\_direction[num\_of\_bullets];

double center\_x;

int score;

extern void renderBitmapString(float x, float y, void \*font, char \*string);

extern void renderVerticalBitmapString(float x, float y, int bitmapHeight, void \*font, char \*string);

void CGameHandler::StartOfGame()

{

score=0;

show=false;

frame\_number=0;

dor=0;

stopmove = true;

rotate\_speed = 0.5;

stop = true;

move=0;

bullets\_number = 0;

//variables for game handling

font = def\_font = (int)GLUT\_BITMAP\_HELVETICA\_18;

ChangeBackgroundColors(2.0/80,90.0/80,155.0/80);

opening.ScalePolygon(10,10,10);

opening.MovePolygon(10,10,10);

opening.SetPolygonColor(0,0,0);

//settings of the objects:

aim.ScalePolygon(0.5,0.5,0);

//cube.MoveModelCenter(-30,15,-51);

aim.MovePolygonCenter(0,0,-19.5);

aim.SetPolygonColor(0,0,0);

plate.MoveModelCenter(-10,3,-20);

plate.RotateSelfModel(X\_AXIS, 90);

plate.ScaleSelfModel(1, 0.2, 1);

cylinder.MoveModel(0, -3, -9.5);

cylinder.ScaleSelfModel(1, 1, 2);

point\_index = 8;

}

//on each frame:

void CGameHandler::DrawScene()

{

if(show==true)

{

plate.DrawModel();

}

/\*circle.DrawPolygon();

circle2.DrawPolygon();

part\_of\_cylinder.DrawPolygon();\*/

cylinder.DrawModel();\

opening.DrawPolygon();

aim.DrawPolygon();

double aim\_z = aim.GetPolygonCenter().GetCoordinate(Z\_AXIS);

for (int bullet\_index = 0; bullet\_index<bullets\_number; bullet\_index++)

{

if (bullets[bullet\_index].GetModelCenter().GetCoordinate(Z\_AXIS) - aim\_z > -0.01)

bullets[bullet\_index].DrawModel();

}

}

void CGameHandler::LogicPart()

{

CPoint plate\_center = plate.GetModelCenter();

double plate\_width = 1;

double plate\_height = 0.2;

double plate\_depth = 1;

double bullet\_width = 0.1;

double bullet\_height = 0.1;

double bullet\_depth = 0.8;

for (int bullet\_index = 0; bullet\_index<bullets\_number; bullet\_index++)

{

if (bullets[bullet\_index].GetModelCenter().GetCoordinate(Z\_AXIS) > -100)

{

bullets[bullet\_index].MoveModel(bullet\_direction[bullet\_index]);

CPoint bullet\_center = bullets[bullet\_index].GetModelCenter();

double z\_diff = abs(bullet\_center.GetCoordinate(Z\_AXIS) - plate\_center.GetCoordinate(Z\_AXIS));

bool hit\_z = z\_diff < (plate\_depth+bullet\_depth)/2;

double x\_diff = abs(bullet\_center.GetCoordinate(X\_AXIS) - plate\_center.GetCoordinate(X\_AXIS));

bool hit\_x = x\_diff < (plate\_width+bullet\_width)/2;

double y\_diff = abs(bullet\_center.GetCoordinate(Y\_AXIS) - plate\_center.GetCoordinate(Y\_AXIS));

bool hit\_y = y\_diff < (plate\_height+bullet\_height)/2;

if (hit\_x && hit\_y && hit\_z)

plate.MoveModelCenter(100,100,100);

score++;

}

}

if(frame\_number>=200)

{

cube.MoveModel(Y\_AXIS , 0.004);

frame\_number--;

}

if(frame\_number<=200&&frame\_number>0)

{

cube.MoveModel(Y\_AXIS ,-0.004);

frame\_number--;

}

if (frame\_number>0 && frame\_number<=400)

{

if(frame\_number>=200)

cube.MoveModel(Y\_AXIS , 0.004);

else

cube.MoveModel(Y\_AXIS , -0.004);

frame\_number--;

}

{

if(stopmove==false)

cube.MoveModel(X\_AXIS , 0);

}

{

if (stop==true)

cube.RotateSelfModel(Y\_AXIS , rotate\_speed);

}

{

if(move>=0.001)

plate.MoveModel(X\_AXIS ,move);

}

{

if(move<=0.001)

plate.MoveModel(X\_AXIS ,move);

}

}

void CGameHandler::WriteText()

{

font = (int) GLUT\_BITMAP\_TIMES\_ROMAN\_24;

glColor3f(100,0,0);

int x\_pos = 10;

int y\_pos = 30;

renderBitmapString(x\_pos, y\_pos, (void \*)font, "Hit The Plate");

font = (int) GLUT\_BITMAP\_TIMES\_ROMAN\_24;

glColor3f(100,0,0);

int x\_pos2 = 10;

int y\_pos2 = 60;

char bullets\_string[20];

if(bullets\_number<num\_of\_bullets)

{

sprintf\_s(bullets\_string, "Bullets : %d", num\_of\_bullets-bullets\_number);

renderBitmapString(x\_pos2,y\_pos2, (void \*)font, bullets\_string);

}

/\*if(score>0)

{

sprintf\_s(bullets\_string, "Score : %d", score);

renderBitmapString(x\_pos2-10,y\_pos2-10, (void \*)font, 10);

}

if(bullets\_number==num\_of\_bullets)

{

font = (int) GLUT\_BITMAP\_TIMES\_ROMAN\_24;

glColor3f(100,0,0);

int x\_pos = 500;

int y\_pos = 30;

renderBitmapString(x\_pos, y\_pos, (void \*)font, "Game Over");

}\*/

if(move==0)

{

font = (int) GLUT\_BITMAP\_TIMES\_ROMAN\_24;

glColor3f(100,0,0);

int x\_pos = 400;

int y\_pos = 20;

renderBitmapString(x\_pos, y\_pos, (void \*)font, "Press 'S' To Start");

}

//char point\_index\_string[30];

//sprintf(point\_index\_string, "point index = %d", point\_index);

//renderBitmapString(x\_pos, y\_pos+30, (void \*)font, point\_index\_string);

}

//keys:

void CGameHandler::KeyPressed(char key)

{

switch (key)

{

case ' '://spacebar

if (bullets\_number < num\_of\_bullets)

{

CPoint aim\_center = aim.GetPolygonCenter();

CPoint canon\_end = cylinder.GetPolygon(cylinder.GetNumberOfPolygons()-1).GetPolygonCenter();

canon\_end.ScalePoint(-1,-1,-1);

bullet\_direction[bullets\_number] = aim\_center;

bullet\_direction[bullets\_number].MovePoint(canon\_end);

bullet\_direction[bullets\_number].ScalePoint(0.05,0.05,0.05);

canon\_end.ScalePoint(-1,-1,-1);

bullets[bullets\_number].SetModelColor(10,0,5);

bullets[bullets\_number].MoveModelCenter(canon\_end);

bullets[bullets\_number].ScaleSelfModel(0.1,0.1,0.8);

bullets\_number++;

}

break;

case 13: //Enter

break;

case 27://Esc

break;

case '1':

break;

case '2':

break;

case '3':

break;

case '4':

break;

case '5':

break;

case 's':

move=0.05;

show=true;

break;

default:

break;

}

}

void CGameHandler::SpecialKeyPressed(char key)

{

switch (key)

{

case GLUT\_KEY\_LEFT:

move-=0.001;

cylinder.RotateSelfModel(Y\_AXIS, 1);

aim.MovePolygon(X\_AXIS , -1);

break;

case GLUT\_KEY\_RIGHT:

aim.MovePolygon(X\_AXIS , 1);

cylinder.RotateSelfModel(Y\_AXIS, -1);

break;

case GLUT\_KEY\_UP:

aim.MovePolygon(Y\_AXIS, 0.5);

cylinder.RotateSelfModel(X\_AXIS, 0.5);

break;

case GLUT\_KEY\_DOWN:

aim.MovePolygon(Y\_AXIS,-0.5);

cylinder.RotateSelfModel(X\_AXIS, -0.5);

break;

default:

break;

}

}

float CGameHandler::GetBeckgroundColorR()

{

return BGColor\_R;

}

float CGameHandler::GetBeckgroundColorG()

{

return BGColor\_G;

}

float CGameHandler::GetBeckgroundColorB()

{

return BGColor\_B;

}

//internal procedures:

void CGameHandler::ChangeBackgroundColors(float R, float G, float B)

{

BGColor\_R = R;

BGColor\_G = G;

BGColor\_B = B;

}

**CModel.h :**

#pragma once

#include "CPoint.h"

#include "CPolygon.h"

class CModel

{

public:

#pragma region CONSTRUCTORs & DESTRUCTOR

CModel();

CModel(int num\_of\_polygons);

CModel(const CModel& model);

~CModel();

#pragma endregion

#pragma region Setting model properties

void SetNumberOfPolygons(int num\_of\_polygons);

void SetModelColor(Color color\_component, double color\_value);

void SetModelColor(double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Setting one polygon properties

void SetOnePolygon(int polygon\_index, CPolygon polygon);

void SetOnePolygonColor(int polygon\_index, Color color\_component, double color\_value);

void SetOnePolygonColor(int polygon\_index, double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Move model functions

void MoveModel(Axis axis, double move\_value);

void MoveModel(double x\_move, double y\_move,double z\_move);

void ScaleSelfModel(double x\_ratio, double y\_ratio, double z\_ratio);

#pragma endregion

#pragma region Rotate model functions

void RotateModel(Axis axis, double rotate\_value);

void RotateSelfModel(Axis axis, double rotate\_value);

#pragma endregion

#pragma region Draw model function

void DrawModel();

#pragma endregion

#pragma region Get functions

CPolygon GetPolygon(int polygon\_index);

CPoint GetModelCenter();

int GetNumberOfPolygons();

#pragma endregion

protected:

#pragma region Private variables

int \_num\_of\_polygons;

CPolygon\* \_polygons\_array;

#pragma endregion

};

**CModel.cpp :**

#include "CModel.h"

#pragma region CONSTRUCTORs & DESTRUCTOR

CModel::CModel()

{

\_num\_of\_polygons = 2;

\_polygons\_array = new CPolygon [\_num\_of\_polygons];

}

CModel::CModel(int num\_of\_polygons)

{

\_num\_of\_polygons= num\_of\_polygons;

\_polygons\_array = new CPolygon [\_num\_of\_polygons];

}

CModel::CModel(const CModel& model)

{

\_num\_of\_polygons = model.\_num\_of\_polygons;

\_polygons\_array = new CPolygon[\_num\_of\_polygons];

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].DuplicatePolygon(model.\_polygons\_array[i]);

}

CModel::~CModel()

{

delete[] \_polygons\_array;

}

#pragma endregion

#pragma region Setting model properties

void CModel::SetNumberOfPolygons(int num\_of\_polygons)

{

delete[] \_polygons\_array;

\_num\_of\_polygons = num\_of\_polygons;

\_polygons\_array = new CPolygon [\_num\_of\_polygons];

}

void CModel::SetModelColor(Color color\_component, double color\_value)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].SetPolygonColor(color\_component, color\_value);

}

void CModel::SetModelColor(double red\_component, double green\_component, double blue\_component)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].SetPolygonColor(red\_component,green\_component,blue\_component);

}

#pragma endregion

#pragma region Setting one polygon properties

void CModel::SetOnePolygon(int polygon\_index, CPolygon polygon)

{

\_polygons\_array[polygon\_index].DuplicatePolygon(polygon);

}

void CModel::SetOnePolygonColor(int polygon\_index, Color color\_component, double color\_value)

{

\_polygons\_array[polygon\_index].SetPolygonColor(color\_component,color\_value);

}

void CModel::SetOnePolygonColor(int polygon\_index, double red\_component, double green\_component, double blue\_component)

{

\_polygons\_array[polygon\_index].SetPolygonColor(red\_component,green\_component,blue\_component);

}

#pragma endregion

#pragma region Move model functions

void CModel::MoveModel(Axis axis, double move\_value)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].MovePolygon(axis,move\_value);

}

void CModel::MoveModel(double x\_move, double y\_move,double z\_move)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].MovePolygon(x\_move,y\_move,z\_move);

}

void CModel::MoveModel(CPoint point\_move)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].MovePolygon(point\_move);

}

void CModel::MoveModelCenter(Axis axis, double coordinate\_value)

{

CPoint p = GetModelCenter();

p.ScalePoint(-1, -1, -1);

MoveModel(p);

MoveModel(axis,coordinate\_value);

}

void CModel::MoveModelCenter(double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

CPoint p = GetModelCenter();

p.ScalePoint(-1, -1, -1);

MoveModel(p);

MoveModel(x\_coordinate, y\_coordinate, z\_coordinate);

}

void CModel::MoveModelCenter(CPoint point\_coordinate)

{

CPoint p = GetModelCenter();

p.ScalePoint(-1, -1, -1);

MoveModel(p);

MoveModel(point\_coordinate);

}

#pragma endregion

#pragma region Scale model functions

void CModel::ScaleModel(Axis axis, double scale\_ratio)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].ScalePolygon(axis,scale\_ratio);

}

void CModel::ScaleModel(double x\_ratio, double y\_ratio, double z\_ratio)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].ScalePolygon(x\_ratio , y\_ratio , z\_ratio);

}

void CModel::ScaleSelfModel(Axis axis, double scale\_ratio)

{

CPoint model\_center = GetModelCenter();

model\_center.ScalePoint(-1, -1, -1);

MoveModel(model\_center);

model\_center.ScalePoint(-1, -1, -1);

ScaleModel(axis, scale\_ratio);

MoveModel(model\_center);

}

void CModel::ScaleSelfModel(double x\_ratio, double y\_ratio, double z\_ratio)

{

CPoint model\_center = GetModelCenter();

model\_center.ScalePoint(-1, -1, -1);

MoveModel(model\_center);

model\_center.ScalePoint(-1, -1, -1);

ScaleModel(x\_ratio,y\_ratio,z\_ratio);

MoveModel(model\_center);

}

#pragma endregion

#pragma region Rotate model functions

void CModel::RotateModel(Axis axis, double rotate\_value)

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].RotatePolygon(axis,rotate\_value);

}

void CModel::RotateSelfModel(Axis axis, double rotate\_value)

{

CPoint model\_center = GetModelCenter();

model\_center.ScalePoint(-1, -1, -1);

MoveModel(model\_center);

model\_center.ScalePoint(-1, -1, -1);

RotateModel(axis, rotate\_value);

MoveModel(model\_center);

}

#pragma endregion

#pragma region Draw model function

void CModel::DrawModel()

{

for (int i = 0; i < \_num\_of\_polygons; i++)

\_polygons\_array[i].DrawPolygon();

}

#pragma endregion

#pragma region Get functions

CPolygon CModel::GetPolygon(int polygon\_index)

{

return \_polygons\_array[polygon\_index];

}

CPoint CModel::GetModelCenter()

{

CPoint sum(0,0,0);

for (int i = 0; i < \_num\_of\_polygons; i++)

sum.MovePoint(\_polygons\_array[i].GetPolygonCenter());

double scale\_ratio = 1.0/\_num\_of\_polygons;

sum.ScalePoint(scale\_ratio, scale\_ratio, scale\_ratio);

return sum;

}

int CModel::GetNumberOfPolygons()

{

return \_num\_of\_polygons;

}

#pragma endregion

**CMyOpenGLInit.h :**

#pragma once

class CMyOpenGLInit

{

public:

CMyOpenGLInit(int argc, char\* argv[]);

void CMyOpenGLInit::InitParameters();

};

**CMyOpenGLInit.cpp :**

#include "CGameHandler.h"

#include "CMyOpenGLInit.h"

#include "glut.h"

CGameHandler MyGame;

void special(int key, int x, int y);

typedef unsigned char ubyte;

void specialkeys( int key, int x, int y );

void keyboard(unsigned char key, int x, int y);

void setOrthographicProjection()

{

/\* switch to projection mode \*/

glMatrixMode(GL\_PROJECTION);

/\* save previous matrix which contains the settings for the perspective projection \*/

glPushMatrix();

/\* reset matrix \*/

glLoadIdentity();

/\* set a 2D orthographic projection \*/

gluOrtho2D(0, glutGet(GLUT\_SCREEN\_WIDTH), 0, glutGet(GLUT\_SCREEN\_HEIGHT));

/\* invert the y axis, down is positive \*/

glScalef(1, -1, 1);

/\* move the origin from the bottom left corner to the upper left corner \*/

glTranslatef(0, -glutGet(GLUT\_SCREEN\_HEIGHT), 0);

glMatrixMode(GL\_MODELVIEW);

}

void resetPerspectiveProjection()

{

glMatrixMode(GL\_PROJECTION);

glPopMatrix();

glMatrixMode(GL\_MODELVIEW);

}

void renderBitmapString(float x, float y, void \*font, char \*string)

{

char \*c;

glRasterPos2f(x, y);

for (c = string; \*c != '\0'; c++)

{

glutBitmapCharacter(font, \*c);

}

}

void renderSpacedBitmapString(float x, float y, int spacing, void \*font, char \*string)

{

char \*c;

int x1 = x;

for (c = string; \*c != '\0'; c++)

{

glRasterPos2f(x1, y);

glutBitmapCharacter(font, \*c);

x1 = x1 + glutBitmapWidth(font, \*c) + spacing;

}

}

void renderVerticalBitmapString(float x, float y, int bitmapHeight, void \*font, char \*string)

{

char \*c;

int i;

for (c = string, i = 0; \*c != '\0'; i++, c++)

{

glRasterPos2f(x, y+bitmapHeight\*i);

glutBitmapCharacter(font, \*c);

}

}

void MyDraw()

{

glClear( GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

glPushMatrix();

glutPostRedisplay();

glClearColor(MyGame.GetBeckgroundColorR(), MyGame.GetBeckgroundColorG(), MyGame.GetBeckgroundColorB(), 0);

MyGame.DrawScene();

MyGame.LogicPart();

glPushMatrix();

glColor3f(0.0f, 0.0f, 1.0f);

setOrthographicProjection();

glLoadIdentity();

MyGame.WriteText();

resetPerspectiveProjection();

glPopMatrix();

glPopMatrix();

glutSwapBuffers();

}

void reshape( GLsizei width, GLsizei height )

{

GLdouble aspect;

glViewport( 0, 0, width, height );

/\* compute aspect ratio \*/

aspect = (GLdouble) width / (GLdouble) height;

glMatrixMode( GL\_PROJECTION );

/\* Reset world coordinates first ... \*/

glLoadIdentity();

/\* Reset the viewing volume based on the new aspect ratio \*/

gluPerspective( 45.0, aspect, 1.0, 300.0 );

glMatrixMode( GL\_MODELVIEW );

}

CMyOpenGLInit::CMyOpenGLInit(int argc, char\* argv[])

{

int x, y;

MyGame.StartOfGame();

glutInit(&argc, argv);

x = glutGet(GLUT\_SCREEN\_WIDTH);

y = glutGet(GLUT\_SCREEN\_HEIGHT);

glutInitWindowPosition(0, 0);

glutInitWindowSize(x, y);

glutInitDisplayMode( GLUT\_RGBA | GLUT\_DEPTH | GLUT\_DOUBLE );

glutCreateWindow("Computer Graphics Project");

InitParameters();

glutKeyboardFunc( keyboard );

glutSpecialFunc( special );

glutReshapeFunc( reshape );

glutDisplayFunc (MyDraw);

glClearColor(MyGame.GetBeckgroundColorR(), MyGame.GetBeckgroundColorG(), MyGame.GetBeckgroundColorB(), 0);

glutMainLoop();

}

void CMyOpenGLInit::InitParameters()

{

/\* enable the depth buffer \*/

glEnable( GL\_DEPTH\_TEST );

/\* enable the face culling \*/

glEnable( GL\_CULL\_FACE );

/\* Turn on the default light \*/

glEnable( GL\_LIGHT0 );

glEnable( GL\_LIGHTING );

/\* have OpenGL automatically normalize the normals, since we have lighting

turned on and are scaling \*/

glEnable( GL\_NORMALIZE );

/\* Enable fast material changes for diffuse material \*/

glColorMaterial( GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE );

glEnable( GL\_COLOR\_MATERIAL );

}

void keyboard(unsigned char key, int x, int y)

{

MyGame.KeyPressed(key);

glutPostRedisplay();

}

void special(int key, int x, int y)

{

MyGame.SpecialKeyPressed(key);

glutPostRedisplay();

}

**Common.h :**

#pragma once

int random(int num);

float Pie();

**Common.cpp :**

#include "Common.h"

#include <stdlib.h>

//global functions:

int random(int num)

{

//you need to put the next line before the random call:

//srand( (unsigned)time( NULL ) );

return int(float(rand())/RAND\_MAX\*num);

}

float Pie()

{

return (float)3.14159;

}

**CPoint.h :**

#pragma once

#pragma region ENUMs

/// Holds the point available axes

enum Axis

{

/// The x axis

X\_AXIS = 1,

/// The y axis

Y\_AXIS = 2,

/// The z axis

Z\_AXIS = 3

};

/// Holds the color's components of a point

enum Color

{

/// The red component of a point

RED\_COMPONENT = 1,

/// The green component of a point

GREEN\_COMPONENT = 2,

/// The blue component of a point

BLUE\_COMPONENT = 3

};

#pragma endregion

class CPoint

{

public:

#pragma region CONSTRUCTORs

/// The basic constructor - creates a new instance of CPoint class

CPoint();

CPoint(double x\_coordinate, double y\_coordinate, double z\_coordinate);

CPoint(double x\_coordinate, double y\_coordinate, double z\_coordinate, double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Move point functions

void MovePoint(Axis axis, double move\_value);

void MovePoint(double x\_move, double y\_move, double z\_move);

void MovePoint(CPoint point\_move);

#pragma endregion

#pragma region Set point location functions

void SetPointCoordinate(Axis axis, double coordinate\_value);

void SetPointCoordinates(double x\_coordinate, double y\_coordinate, double z\_coordinate);

#pragma endregion

#pragma region Scale point functions

void ScalePoint(Axis axis, double scale\_ratio);

void ScalePoint(double x\_ratio, double y\_ratio, double z\_ratio);

#pragma endregion

#pragma region Rotate point functions

void RotatePoint(Axis axis, double rotate\_value);

#pragma endregion

#pragma region Set point color functions

void SetPointColor(Color color\_component, double color\_value);

void SetPointColor(double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Get functions

CPoint ClonePoint();

double GetCoordinate(Axis axis);

double GetColor(Color color\_component);

#pragma endregion

#pragma region Draw function

void DrawPoint();

#pragma endregion

private:

#pragma region Private variables

double \_x\_coordinate;

double \_y\_coordinate;

double \_z\_coordinate;

double \_red\_component;

double \_green\_component;

double \_blue\_component;

#pragma endregion

#pragma region Move point functions

void MovePointX(double x\_move);

void MovePointY(double y\_move);

void MovePointZ(double z\_move);

#pragma endregion

#pragma region Set point location functions

void SetPointX(double x\_coordinate);

void SetPointY(double y\_coordinate);

void SetPointZ(double z\_coordinate);

#pragma endregion

#pragma region Rotate point functions

void RotatePointX(double x\_rotate);

void RotatePointY(double y\_rotate);

void RotatePointZ(double z\_rotate);

#pragma endregion

#pragma region Set point color functions

void SetPointRed(double red\_component);

void SetPointGreen(double green\_component);

void SetPointBlue(double blue\_component);

#pragma endregion

#pragma region Get functions

double GetX();

double GetY();

double GetZ();

double GetRed();

double GetGreen();

double GetBlue();

#pragma endregion

};

**CPoint.cpp :**

#include <math.h>

#include "CPoint.h"

#include "glut.h"

const float PI = (float)3.1415926536;

#pragma region Constructors

CPoint::CPoint()

{

\_x\_coordinate = 0;

\_y\_coordinate = 0;

\_z\_coordinate = 0;

\_red\_component = 0;

\_green\_component = 0;

\_blue\_component = 0;

}

CPoint::CPoint(double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

\_x\_coordinate = x\_coordinate;

\_y\_coordinate = y\_coordinate;

\_z\_coordinate = z\_coordinate;

\_red\_component = 0;

\_green\_component = 0;

\_blue\_component = 0;

}

CPoint::CPoint(double x\_coordinate, double y\_coordinate, double z\_coordinate, double r\_color, double g\_color, double b\_color)

{

\_x\_coordinate = x\_coordinate;

\_y\_coordinate = y\_coordinate;

\_z\_coordinate = z\_coordinate;

\_red\_component = r\_color;

\_green\_component = g\_color;

\_blue\_component = b\_color;

}

#pragma endregion

#pragma region Move point functions

void CPoint::MovePoint(Axis axis, double move\_value)

{

switch (axis)

{

case X\_AXIS:

MovePointX(move\_value);

break;

case Y\_AXIS:

MovePointY(move\_value);

break;

case Z\_AXIS:

MovePointZ(move\_value);

break;

}

}

void CPoint::MovePointX(double x\_move)

{

\_x\_coordinate += x\_move;

}

void CPoint::MovePointY(double y\_move)

{

\_y\_coordinate += y\_move;

}

void CPoint::MovePointZ(double z\_move)

{

\_z\_coordinate += z\_move;

}

void CPoint::MovePoint(double x\_move, double y\_move, double z\_move)

{

MovePointX(x\_move);

MovePointY(y\_move);

MovePointZ(z\_move);

}

void CPoint::MovePoint(CPoint point\_move)

{

MovePointX(point\_move.GetX());

MovePointY(point\_move.GetY());

MovePointZ(point\_move.GetZ());

}

#pragma endregion

#pragma region Set point location functions

void CPoint::SetPointCoordinate(Axis axis, double coordinate\_value)

{

switch (axis)

{

case X\_AXIS:

SetPointX(coordinate\_value);

break;

case Y\_AXIS:

SetPointY(coordinate\_value);

break;

case Z\_AXIS:

SetPointZ(coordinate\_value);

break;

}

}

void CPoint::SetPointX(double x\_coordinate)

{

\_x\_coordinate = x\_coordinate;

}

void CPoint::SetPointY(double y\_coordinate)

{

\_y\_coordinate = y\_coordinate;

}

void CPoint::SetPointZ(double z\_coordinate)

{

\_z\_coordinate = z\_coordinate;

}

void CPoint::SetPointCoordinates(double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

SetPointX(x\_coordinate);

SetPointY(y\_coordinate);

SetPointZ(z\_coordinate);

}

#pragma endregion

#pragma region Scale point functions

void CPoint::ScalePoint(Axis axis, double scale\_ratio)

{

switch (axis)

{

case X\_AXIS:

\_x\_coordinate \*= scale\_ratio;

break;

case Y\_AXIS:

\_y\_coordinate \*= scale\_ratio;

break;

case Z\_AXIS:

\_z\_coordinate \*= scale\_ratio;

break;

}

}

void CPoint::ScalePoint(double x\_ratio, double y\_ratio, double z\_ratio)

{

\_x\_coordinate \*= x\_ratio;

\_y\_coordinate \*= y\_ratio;

\_z\_coordinate \*= z\_ratio;

}

#pragma endregion

#pragma region Rotate point functions

void CPoint::RotatePoint(Axis axis, double rotate\_value)

{

switch (axis)

{

case X\_AXIS:

RotatePointX(rotate\_value);

break;

case Y\_AXIS:

RotatePointY(rotate\_value);

break;

case Z\_AXIS:

RotatePointZ(rotate\_value);

break;

}

}

void CPoint::RotatePointX(double x\_rotate)

{

double angle\_in\_radians = x\_rotate \* PI / 180;

double original\_y\_coordinate = \_y\_coordinate;

double cos\_value = cos(angle\_in\_radians);

double sin\_value = sin(angle\_in\_radians);

\_y\_coordinate = original\_y\_coordinate\*cos\_value - \_z\_coordinate\*sin\_value;

\_z\_coordinate = original\_y\_coordinate\*sin\_value + \_z\_coordinate\*cos\_value;

}

void CPoint::RotatePointY(double y\_rotate)

{

double angle\_in\_radians = y\_rotate \* PI/180;

double original\_x\_coordinate = \_x\_coordinate;

double cos\_value = cos(angle\_in\_radians);

double sin\_value = sin(angle\_in\_radians);

\_x\_coordinate = \_z\_coordinate\*sin\_value + original\_x\_coordinate\*cos\_value;

\_z\_coordinate = \_z\_coordinate\*cos\_value - original\_x\_coordinate\*sin\_value;

}

void CPoint::RotatePointZ(double z\_rotate)

{

double angle\_in\_radians = z\_rotate \* PI/180;

double original\_x\_coordinate = \_x\_coordinate;

double cos\_value = cos(angle\_in\_radians);

double sin\_value = sin(angle\_in\_radians);

\_x\_coordinate = original\_x\_coordinate\*cos\_value - \_y\_coordinate\*sin\_value;

\_y\_coordinate = original\_x\_coordinate\*sin\_value + \_y\_coordinate\*cos\_value;

}

#pragma endregion

#pragma region Set point color functions

void CPoint::SetPointColor(Color color, double color\_value)

{

switch (color)

{

case RED\_COMPONENT:

SetPointRed(color\_value);

break;

case GREEN\_COMPONENT:

SetPointGreen(color\_value);

break;

case BLUE\_COMPONENT:

SetPointBlue(color\_value);

break;

}

}

void CPoint::SetPointRed(double r\_color)

{

\_red\_component = r\_color;

}

void CPoint::SetPointGreen(double g\_color)

{

\_green\_component = g\_color;

}

void CPoint::SetPointBlue(double b\_color)

{

\_blue\_component = b\_color;

}

void CPoint::SetPointColor(double r\_color, double g\_color, double b\_color)

{

SetPointRed(r\_color);

SetPointGreen(g\_color);

SetPointBlue(b\_color);

}

#pragma endregion

#pragma region Get functions

CPoint CPoint::ClonePoint()

{

CPoint new\_point(GetX(), GetY(), GetZ(), GetRed(), GetGreen(), GetBlue());

return new\_point;

}

double CPoint::GetCoordinate(Axis axis)

{

switch (axis)

{

case X\_AXIS:

return GetX();

break;

case Y\_AXIS:

return GetY();

break;

case Z\_AXIS:

return GetZ();

break;

default:

return 0;

}

}

double CPoint::GetColor(Color color)

{

switch (color)

{

case RED\_COMPONENT:

return GetRed();

break;

case GREEN\_COMPONENT:

return GetGreen();

break;

case BLUE\_COMPONENT:

return GetBlue();

break;

default:

return -1;

}

}

double CPoint::GetX()

{

return \_x\_coordinate;

}

double CPoint::GetY()

{

return \_y\_coordinate;

}

double CPoint::GetZ()

{

return \_z\_coordinate;

}

double CPoint::GetRed()

{

return \_red\_component;

}

double CPoint::GetGreen()

{

return \_green\_component;

}

double CPoint::GetBlue()

{

return \_blue\_component;

}

#pragma endregion

#pragma region Draw function

void CPoint::DrawPoint()

{

glColor3f(\_red\_component, \_green\_component, \_blue\_component);

glVertex3f(\_x\_coordinate, \_y\_coordinate, \_z\_coordinate);

}

#pragma endregion

**CPolygon.h :**

#pragma once

#include "CPoint.h"

class CPolygon

{

public:

#pragma region CONSTRUCTORs & DESTRUCTOR

CPolygon();

CPolygon(int num\_of\_points);

CPolygon(const CPolygon& polygon);

~CPolygon();

#pragma endregion

#pragma region Setting polygon properties

void SetNumberOfPoints(int num\_of\_points);

void SetPolygonColor(Color color\_component, double color\_value);

void SetPolygonColor(double red\_component, double green\_component, double blue\_component);

void DuplicatePolygon(CPolygon polygon);

#pragma endregion

#pragma region Setting one point properties

void SetOnePoint(int point\_index, CPoint point);

void SetOnePointCoordinates(int point\_index, double x\_coordinate, double y\_coordinate, double z\_coordinate);

void SetOnePointColor(int point\_index, Color color\_component, double color\_value);

void SetOnePointColor(int point\_index, double red\_component, double green\_component, double blue\_component);

#pragma endregion

#pragma region Move polygon functions

void MovePolygon(Axis axis, double move\_value);

void MovePolygon(double x\_move, double y\_move,double z\_move);

void MovePolygon(CPoint point\_move);

void MovePolygonCenter(Axis axis, double coordinate\_value);

void MovePolygonCenter(double x\_coordinate, double y\_coordinate,double z\_coordinate);

void MovePolygonCenter(CPoint point\_coordinate);

#pragma endregion

#pragma region Scale polygon functions

void ScalePolygon(Axis axis, double scale\_ratio);

void ScalePolygon(double x\_ratio, double y\_ratio, double z\_ratio);

void ScaleSelfPolygon(Axis axis, double scale\_ratio);

void ScaleSelfPolygon(double x\_ratio, double y\_ratio, double z\_ratio);

#pragma endregion

#pragma region Rotate polygon functions

void RotatePolygon(Axis axis, double rotate\_value);

void RotateSelfPolygon(Axis axis, double rotate\_value);

#pragma endregion

#pragma region Draw polygon function

void DrawPolygon();

#pragma endregion

#pragma region Get functions

CPoint GetPoint(int point\_index);

CPoint GetPolygonCenter();

int GetNumberOfPoints();

#pragma endregion

private:

#pragma region Private variables

int \_num\_of\_points;

CPoint\* \_points\_array;

#pragma endregion

};

**CPolygon.cpp :**

#include <stdlib.h>

#include "glut.h"

#include "CPolygon.h"

#pragma region CONSTRUCTORs & DESTRUCTOR

CPolygon::CPolygon()

{

\_num\_of\_points = 2;

\_points\_array = new CPoint[\_num\_of\_points];

}

CPolygon::CPolygon(int num\_of\_points)

{

\_num\_of\_points = num\_of\_points;

\_points\_array = new CPoint[\_num\_of\_points];

}

CPolygon::CPolygon(const CPolygon& polygon)

{

\_num\_of\_points = polygon.\_num\_of\_points;

\_points\_array = new CPoint[\_num\_of\_points];

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i] = polygon.\_points\_array[i].ClonePoint();

}

CPolygon::~CPolygon()

{

delete[] \_points\_array;

}

#pragma endregion

#pragma region Setting CPolygon properties

void CPolygon::SetNumberOfPoints(int num\_of\_points)

{

\_num\_of\_points = num\_of\_points;

delete[] \_points\_array;

\_points\_array = new CPoint[\_num\_of\_points];

}

void CPolygon::SetPolygonColor(Color color\_component, double color\_value)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].SetPointColor(color\_component, color\_value);

}

void CPolygon::SetPolygonColor(double red\_component, double green\_component, double blue\_component)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].SetPointColor(red\_component, green\_component, blue\_component);

}

void CPolygon::DuplicatePolygon(CPolygon polygon)

{

SetNumberOfPoints(polygon.GetNumberOfPoints());

for (int i = 0; i < \_num\_of\_points; i++)

SetOnePoint(i, polygon.GetPoint(i));

}

#pragma endregion

#pragma region Setting one point properties

void CPolygon::SetOnePoint(int point\_index, CPoint point)

{

\_points\_array[point\_index] = point.ClonePoint();

}

void CPolygon::SetOnePointCoordinates(int point\_index, double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

\_points\_array[point\_index].SetPointCoordinates(x\_coordinate, y\_coordinate, z\_coordinate);

}

void CPolygon::SetOnePointColor(int point\_index, Color color\_component, double color\_value)

{

\_points\_array[point\_index].SetPointColor(color\_component, color\_value);

}

void CPolygon::SetOnePointColor(int point\_index, double red\_component, double green\_component, double blue\_component)

{

\_points\_array[point\_index].SetPointColor(red\_component, green\_component, blue\_component);

}

#pragma endregion

#pragma region Move polygon functions

void CPolygon::MovePolygon(Axis axis, double move\_value)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].MovePoint(axis, move\_value);

}

void CPolygon::MovePolygon(double x\_move, double y\_move, double z\_move)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].MovePoint(x\_move, y\_move, z\_move);

}

void CPolygon::MovePolygon(CPoint point\_move)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].MovePoint(point\_move);

}

void CPolygon::MovePolygonCenter(Axis axis, double coordinate\_value)

{

CPoint p = GetPolygonCenter();

MovePolygon(axis, coordinate\_value - p.GetCoordinate(axis));

}

void CPolygon::MovePolygonCenter(double x\_coordinate, double y\_coordinate, double z\_coordinate)

{

CPoint p = GetPolygonCenter();

p.ScalePoint(-1, -1, -1);

MovePolygon(p);

MovePolygon(x\_coordinate, y\_coordinate, z\_coordinate);

}

void CPolygon::MovePolygonCenter(CPoint point\_coordinate)

{

CPoint p = GetPolygonCenter();

p.ScalePoint(-1, -1, -1);

MovePolygon(p);

MovePolygon(point\_coordinate);

}

#pragma endregion

#pragma region Scale polygon functions

void CPolygon::ScalePolygon(Axis axis, double scale\_ratio)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].ScalePoint(axis, scale\_ratio);

}

void CPolygon::ScalePolygon(double x\_ratio, double y\_ratio, double z\_ratio)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].ScalePoint(x\_ratio, y\_ratio, z\_ratio);

}

void CPolygon::ScaleSelfPolygon(Axis axis, double scale\_ratio)

{

CPoint polygon\_center = GetPolygonCenter();

polygon\_center.ScalePoint(-1, -1, -1);

MovePolygon(polygon\_center);

polygon\_center.ScalePoint(-1, -1, -1);

ScalePolygon(axis, scale\_ratio);

MovePolygon(polygon\_center);

}

void CPolygon::ScaleSelfPolygon(double x\_ratio, double y\_ratio, double z\_ratio)

{

CPoint polygon\_center = GetPolygonCenter();

polygon\_center.ScalePoint(-1, -1, -1);

MovePolygon(polygon\_center);

polygon\_center.ScalePoint(-1, -1, -1);

ScalePolygon(x\_ratio, y\_ratio, z\_ratio);

MovePolygon(polygon\_center);

}

#pragma endregion

#pragma region Rotate polygon functions

void CPolygon::RotatePolygon(Axis axis, double rotate\_value)

{

for (int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].RotatePoint(axis, rotate\_value);

}

void CPolygon::RotateSelfPolygon(Axis axis, double rotate\_value)

{

CPoint polygon\_center = GetPolygonCenter();

polygon\_center.ScalePoint(-1, -1, -1);

MovePolygon(polygon\_center);

polygon\_center.ScalePoint(-1, -1, -1);

RotatePolygon(axis, rotate\_value);

MovePolygon(polygon\_center);

}

#pragma endregion

#pragma region Draw polygon function

void CPolygon::DrawPolygon()

{

glPolygonMode(GL\_FRONT, GL\_FILL);

glBegin(GL\_POLYGON);

for(int i = 0; i < \_num\_of\_points; i++)

\_points\_array[i].DrawPoint();

glEnd();

}

#pragma endregion

#pragma region Get functions

CPoint CPolygon::GetPoint(int point\_index)

{

return \_points\_array[point\_index];

}

CPoint CPolygon::GetPolygonCenter()

{

CPoint sum(0, 0, 0);

for (int i = 0; i < \_num\_of\_points; i++)

sum.MovePoint(\_points\_array[i]);

double scale\_ratio = 1.0/\_num\_of\_points;

sum.ScalePoint(scale\_ratio, scale\_ratio, scale\_ratio);

return sum;

}

int CPolygon::GetNumberOfPoints()

{

return \_num\_of\_points;

}

#pragma endregion

**CSquare.h :**

#pragma once

#include "CPolygon.h"

/// Create black square, from 4 points - (0,0,0); (1,0,0); (1,1,0); (0,1,0)

class CSquare : public CPolygon

{

public:

CSquare(void);

};

**CSquare.cpp :**

#include "CSquare.h"

CSquare::CSquare(void)

{

SetNumberOfPoints(4);

SetOnePointCoordinates(0, 0, 0, 0);

SetOnePointCoordinates(1, 1, 0, 0);

SetOnePointCoordinates(2, 1, 1, 0);

SetOnePointCoordinates(3, 0, 1, 0);

SetPolygonColor(0, 0, 0);

}

**glut.h :**

#ifndef \_\_glut\_h\_\_

#define \_\_glut\_h\_\_

#if defined(\_WIN32)

# if 0

/\* This would put tons of macros and crap in our clean name space. \*/

# define WIN32\_LEAN\_AND\_MEAN

# include <windows.h>

# else

/\* XXX This is from Win32's <windef.h> \*/

# ifndef APIENTRY

# define GLUT\_APIENTRY\_DEFINED

# if (\_MSC\_VER >= 800) || defined(\_STDCALL\_SUPPORTED) || defined(\_\_BORLANDC\_\_) || defined(\_\_LCC\_\_)

# define APIENTRY \_\_stdcall

# else

# define APIENTRY

# endif

# endif

/\* XXX This is from Win32's <winnt.h> \*/

# ifndef CALLBACK

# if (defined(\_M\_MRX000) || defined(\_M\_IX86) || defined(\_M\_ALPHA) || defined(\_M\_PPC)) && !defined(MIDL\_PASS) || defined(\_\_LCC\_\_)

# define CALLBACK \_\_stdcall

# else

# define CALLBACK

# endif

# endif

/\* XXX Hack for lcc compiler. It doesn't support \_\_declspec(dllimport), just \_\_stdcall. \*/

# if defined( \_\_LCC\_\_ )

# undef WINGDIAPI

# define WINGDIAPI \_\_stdcall

# else

/\* XXX This is from Win32's <wingdi.h> and <winnt.h> \*/

# ifndef WINGDIAPI

# define GLUT\_WINGDIAPI\_DEFINED

# define WINGDIAPI \_\_declspec(dllimport)

# endif

# endif

/\* XXX This is from Win32's <ctype.h> \*/

# ifndef \_WCHAR\_T\_DEFINED

typedef unsigned short wchar\_t;

# define \_WCHAR\_T\_DEFINED

# endif

# endif

# if !defined(GLUT\_BUILDING\_LIB) && !defined(GLUT\_NO\_LIB\_PRAGMA)

# pragma comment (lib, "winmm.lib

# ifdef GLUT\_USE\_SGI\_OPENGL

# pragma comment (lib, "opengl.lib") /\* link with SGI OpenGL for Windows lib \*/

# pragma comment (lib, "glu.lib") /\* link with SGI OpenGL Utility lib \*/

# pragma comment (lib, "glut.lib") /\* link with Win32 GLUT for SGI OpenGL lib \*/

# else

# pragma comment (lib, "opengl32.lib") /\* link with Microsoft OpenGL lib \*/

# pragma comment (lib, "glu32.lib") /\* link with Microsoft OpenGL Utility lib \*/

# pragma comment (lib, "glut32.lib") /\* link with Win32 GLUT lib \*/

# endif

# endif

/\* To disable supression of annoying warnings about floats being promoted

to doubles, define GLUT\_NO\_WARNING\_DISABLE in your compile preprocessor

options. \*/

# ifndef GLUT\_NO\_WARNING\_DISABLE

# pragma warning (disable:4244) /\* Disable bogus VC++ 4.2 conversion warnings. \*/

# pragma warning (disable:4305) /\* VC++ 5.0 version of above warning. \*/

# endif

/\* Win32 has an annoying issue where there are multiple C run-time

libraries (CRTs). If the executable is linked with a different CRT

from the GLUT DLL, the GLUT DLL will not share the same CRT static

data seen by the executable. In particular, atexit callbacks registered

in the executable will not be called if GLUT calls its (different)

exit routine). GLUT is typically built with the

"/MD" option (the CRT with multithreading DLL support), but the Visual

C++ linker default is "/ML" (the single threaded CRT).

One workaround to this issue is requiring users to always link with

the same CRT as GLUT is compiled with. That requires users supply a

# if !defined(\_MSC\_VER) && !defined(\_\_cdecl)

/\* Define \_\_cdecl for non-Microsoft compilers. \*/

# define \_\_cdecl

# define GLUT\_DEFINED\_\_\_CDECL

# endif

# ifndef \_CRTIMP

# ifdef \_NTSDK

/\* Definition compatible with NT SDK \*/

# define \_CRTIMP

# else

/\* Current definition \*/

# ifdef \_DLL

# define \_CRTIMP \_\_declspec(dllimport)

# else

# define \_CRTIMP

# endif

# endif

# define GLUT\_DEFINED\_\_CRTIMP

# endif

/\* GLUT API entry point declarations for Win32. \*/

# ifdef GLUT\_BUILDING\_LIB

# define GLUTAPI \_\_declspec(dllexport)

# else

# ifdef \_DLL

# define GLUTAPI \_\_declspec(dllimport)

# else

# define GLUTAPI extern

# endif

# endif

/\* GLUT callback calling convention for Win32. \*/

# define GLUTCALLBACK \_\_cdecl

#endif /\* \_WIN32 \*/

#include <GL/gl.h>

#include <GL/glu.h>

#ifdef \_\_cplusplus

extern "C" {

#endif

#if defined(\_WIN32)

# ifndef GLUT\_BUILDING\_LIB

extern \_CRTIMP void \_\_cdecl exit(int);

# endif

#else

/\* non-Win32 case. \*/

/\* Define APIENTRY and CALLBACK to nothing if we aren't on Win32. \*/

# define APIENTRY

# define GLUT\_APIENTRY\_DEFINED

# define CALLBACK

/\* Define GLUTAPI and GLUTCALLBACK as below if we aren't on Win32. \*/

# define GLUTAPI extern

# define GLUTCALLBACK

/\* Prototype exit for the non-Win32 case (see above). \*/

extern void exit(int);

#endif

/\*\*

GLUT API revision history:

GLUT\_API\_VERSION is updated to reflect incompatible GLUT

API changes (interface changes, semantic changes, deletions,

or additions).

GLUT\_API\_VERSION=1 First public release of GLUT. 11/29/94

GLUT\_API\_VERSION=2 Added support for OpenGL/GLX multisampling,

extension. Supports new input devices like tablet, dial and button

box, and Spaceball. Easy to query OpenGL extensions.

GLUT\_API\_VERSION=3 glutMenuStatus added.

GLUT\_API\_VERSION=4 glutInitDisplayString, glutWarpPointer,

glutBitmapLength, glutStrokeLength, glutWindowStatusFunc, dynamic

video resize subAPI, glutPostWindowRedisplay, glutKeyboardUpFunc,

glutSpecialUpFunc, glutIgnoreKeyRepeat, glutSetKeyRepeat,

glutJoystickFunc, glutForceJoystickFunc (NOT FINALIZED!).

\*\*/

#ifndef GLUT\_API\_VERSION /\* allow this to be overriden \*/

#define GLUT\_API\_VERSION 3

#endif

/\*\*

GLUT implementation revision history:

GLUT\_XLIB\_IMPLEMENTATION is updated to reflect both GLUT

API revisions and implementation revisions (ie, bug fixes).

GLUT\_XLIB\_IMPLEMENTATION=1 mjk's first public release of

GLUT Xlib-based implementation. 11/29/94

GLUT\_XLIB\_IMPLEMENTATION=2 mjk's second public release of

GLUT Xlib-based implementation providing GLUT version 2

interfaces.

GLUT\_XLIB\_IMPLEMENTATION=3 mjk's GLUT 2.2 images. 4/17/95

GLUT\_XLIB\_IMPLEMENTATION=4 mjk's GLUT 2.3 images. 6/?/95

GLUT\_XLIB\_IMPLEMENTATION=5 mjk's GLUT 3.0 images. 10/?/95

GLUT\_XLIB\_IMPLEMENTATION=7 mjk's GLUT 3.1+ with glutWarpPoitner. 7/24/96

GLUT\_XLIB\_IMPLEMENTATION=8 mjk's GLUT 3.1+ with glutWarpPoitner

and video resize. 1/3/97

GLUT\_XLIB\_IMPLEMENTATION=9 mjk's GLUT 3.4 release with early GLUT 4 routines.

GLUT\_XLIB\_IMPLEMENTATION=11 Mesa 2.5's GLUT 3.6 release.

GLUT\_XLIB\_IMPLEMENTATION=12 mjk's GLUT 3.6 release with early GLUT 4 routines + signal handling.

GLUT\_XLIB\_IMPLEMENTATION=13 mjk's GLUT 3.7 beta with GameGLUT support.

GLUT\_XLIB\_IMPLEMENTATION=14 mjk's GLUT 3.7 beta with f90gl friend interface.

GLUT\_XLIB\_IMPLEMENTATION=15 mjk's GLUT 3.7 beta sync'ed with Mesa <GL/glut.h>

\*\*/

#ifndef GLUT\_XLIB\_IMPLEMENTATION /\* Allow this to be overriden. \*/

#define GLUT\_XLIB\_IMPLEMENTATION 15

#endif

/\* Display mode bit masks. \*/

#define GLUT\_RGB 0

#define GLUT\_RGBA GLUT\_RGB

#define GLUT\_INDEX 1

#define GLUT\_SINGLE 0

#define GLUT\_DOUBLE 2

#define GLUT\_ACCUM 4

#define GLUT\_ALPHA 8

#define GLUT\_DEPTH 16

#define GLUT\_STENCIL 32

#if (GLUT\_API\_VERSION >= 2)

#define GLUT\_MULTISAMPLE 128

#define GLUT\_STEREO 256

#endif

#if (GLUT\_API\_VERSION >= 3)

#define GLUT\_LUMINANCE 512

#endif

/\* Mouse buttons. \*/

#define GLUT\_LEFT\_BUTTON 0

#define GLUT\_MIDDLE\_BUTTON 1

#define GLUT\_RIGHT\_BUTTON 2

/\* Mouse button state. \*/

#define GLUT\_DOWN 0

#define GLUT\_UP 1

#if (GLUT\_API\_VERSION >= 2)

/\* function keys \*/

#define GLUT\_KEY\_F1 1

#define GLUT\_KEY\_F2 2

#define GLUT\_KEY\_F3 3

#define GLUT\_KEY\_F4 4

#define GLUT\_KEY\_F5 5

#define GLUT\_KEY\_F6 6

#define GLUT\_KEY\_F7 7

#define GLUT\_KEY\_F8 8

#define GLUT\_KEY\_F9 9

#define GLUT\_KEY\_F10 10

#define GLUT\_KEY\_F11 11

#define GLUT\_KEY\_F12 12

/\* directional keys \*/

#define GLUT\_KEY\_LEFT 100

#define GLUT\_KEY\_UP 101

#define GLUT\_KEY\_RIGHT 102

#define GLUT\_KEY\_DOWN 103

#define GLUT\_KEY\_PAGE\_UP 104

#define GLUT\_KEY\_PAGE\_DOWN 105

#define GLUT\_KEY\_HOME 106

#define GLUT\_KEY\_END 107

#define GLUT\_KEY\_INSERT 108

#endif

/\* Entry/exit state. \*/

#define GLUT\_LEFT 0

#define GLUT\_ENTERED 1

/\* Menu usage state. \*/

#define GLUT\_MENU\_NOT\_IN\_USE 0

#define GLUT\_MENU\_IN\_USE 1

/\* Visibility state. \*/

#define GLUT\_NOT\_VISIBLE 0

#define GLUT\_VISIBLE 1

/\* Window status state. \*/

#define GLUT\_HIDDEN 0

#define GLUT\_FULLY\_RETAINED 1

#define GLUT\_PARTIALLY\_RETAINED 2

#define GLUT\_FULLY\_COVERED 3

/\* Color index component selection values. \*/

#define GLUT\_RED 0

#define GLUT\_GREEN 1

#define GLUT\_BLUE 2

#if defined(\_WIN32)

/\* Stroke font constants (use these in GLUT program). \*/

#define GLUT\_STROKE\_ROMAN ((void\*)0)

#define GLUT\_STROKE\_MONO\_ROMAN ((void\*)1)

/\* Bitmap font constants (use these in GLUT program). \*/

#define GLUT\_BITMAP\_9\_BY\_15 ((void\*)2)

#define GLUT\_BITMAP\_8\_BY\_13 ((void\*)3)

#define GLUT\_BITMAP\_TIMES\_ROMAN\_10 ((void\*)4)

#define GLUT\_BITMAP\_TIMES\_ROMAN\_24 ((void\*)5)

#if (GLUT\_API\_VERSION >= 3)

#define GLUT\_BITMAP\_HELVETICA\_10 ((void\*)6)

#define GLUT\_BITMAP\_HELVETICA\_12 ((void\*)7)

#define GLUT\_BITMAP\_HELVETICA\_18 ((void\*)8)

#endif

#else

/\* Stroke font opaque addresses (use constants instead in source code). \*/

GLUTAPI void \*glutStrokeRoman;

GLUTAPI void \*glutStrokeMonoRoman;

/\* Stroke font constants (use these in GLUT program). \*/

#define GLUT\_STROKE\_ROMAN (&glutStrokeRoman)

#define GLUT\_STROKE\_MONO\_ROMAN (&glutStrokeMonoRoman)

/\* Bitmap font opaque addresses (use constants instead in source code). \*/

GLUTAPI void \*glutBitmap9By15;

GLUTAPI void \*glutBitmap8By13;

GLUTAPI void \*glutBitmapTimesRoman10;

GLUTAPI void \*glutBitmapTimesRoman24;

GLUTAPI void \*glutBitmapHelvetica10;

GLUTAPI void \*glutBitmapHelvetica12;

GLUTAPI void \*glutBitmapHelvetica18;

/\* Bitmap font constants (use these in GLUT program). \*/

#define GLUT\_BITMAP\_9\_BY\_15 (&glutBitmap9By15)

#define GLUT\_BITMAP\_8\_BY\_13 (&glutBitmap8By13)

#define GLUT\_BITMAP\_TIMES\_ROMAN\_10 (&glutBitmapTimesRoman10)

#define GLUT\_BITMAP\_TIMES\_ROMAN\_24 (&glutBitmapTimesRoman24)

#if (GLUT\_API\_VERSION >= 3)

#define GLUT\_BITMAP\_HELVETICA\_10 (&glutBitmapHelvetica10)

#define GLUT\_BITMAP\_HELVETICA\_12 (&glutBitmapHelvetica12)

#define GLUT\_BITMAP\_HELVETICA\_18 (&glutBitmapHelvetica18)

#endif

#endif

/\* glutGet parameters. \*/

#define GLUT\_WINDOW\_X ((GLenum) 100)

#define GLUT\_WINDOW\_Y ((GLenum) 101)

#define GLUT\_WINDOW\_WIDTH ((GLenum) 102)

#define GLUT\_WINDOW\_HEIGHT ((GLenum) 103)

#define GLUT\_WINDOW\_BUFFER\_SIZE ((GLenum) 104)

#define GLUT\_WINDOW\_STENCIL\_SIZE ((GLenum) 105)

#define GLUT\_WINDOW\_DEPTH\_SIZE ((GLenum) 106)

#define GLUT\_WINDOW\_RED\_SIZE ((GLenum) 107)

#define GLUT\_WINDOW\_GREEN\_SIZE ((GLenum) 108)

#define GLUT\_WINDOW\_BLUE\_SIZE ((GLenum) 109)

#define GLUT\_WINDOW\_ALPHA\_SIZE ((GLenum) 110)

#define GLUT\_WINDOW\_ACCUM\_RED\_SIZE ((GLenum) 111)

#define GLUT\_WINDOW\_ACCUM\_GREEN\_SIZE ((GLenum) 112)

#define GLUT\_WINDOW\_ACCUM\_BLUE\_SIZE ((GLenum) 113)

#define GLUT\_WINDOW\_ACCUM\_ALPHA\_SIZE ((GLenum) 114)

#define GLUT\_WINDOW\_DOUBLEBUFFER ((GLenum) 115)

#define GLUT\_WINDOW\_RGBA ((GLenum) 116)

#define GLUT\_WINDOW\_PARENT ((GLenum) 117)

#define GLUT\_WINDOW\_NUM\_CHILDREN ((GLenum) 118)

#define GLUT\_WINDOW\_COLORMAP\_SIZE ((GLenum) 119)

#if (GLUT\_API\_VERSION >= 2)

#define GLUT\_WINDOW\_NUM\_SAMPLES ((GLenum) 120)

#define GLUT\_WINDOW\_STEREO ((GLenum) 121)

#endif

#if (GLUT\_API\_VERSION >= 3)

#define GLUT\_WINDOW\_CURSOR ((GLenum) 122)

#endif

#define GLUT\_SCREEN\_WIDTH ((GLenum) 200)

#define GLUT\_SCREEN\_HEIGHT ((GLenum) 201)

#define GLUT\_SCREEN\_WIDTH\_MM ((GLenum) 202)

#define GLUT\_SCREEN\_HEIGHT\_MM ((GLenum) 203)

#define GLUT\_MENU\_NUM\_ITEMS ((GLenum) 300)

#define GLUT\_DISPLAY\_MODE\_POSSIBLE ((GLenum) 400)

#define GLUT\_INIT\_WINDOW\_X ((GLenum) 500)

#define GLUT\_INIT\_WINDOW\_Y ((GLenum) 501)

#define GLUT\_INIT\_WINDOW\_WIDTH ((GLenum) 502)

#define GLUT\_INIT\_WINDOW\_HEIGHT ((GLenum) 503)

#define GLUT\_INIT\_DISPLAY\_MODE ((GLenum) 504)

#if (GLUT\_API\_VERSION >= 2)

#define GLUT\_ELAPSED\_TIME ((GLenum) 700)

#endif

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 13)

#define GLUT\_WINDOW\_FORMAT\_ID ((GLenum) 123)

#endif

#if (GLUT\_API\_VERSION >= 2)

/\* glutDeviceGet parameters. \*/

#define GLUT\_HAS\_KEYBOARD ((GLenum) 600)

#define GLUT\_HAS\_MOUSE ((GLenum) 601)

#define GLUT\_HAS\_SPACEBALL ((GLenum) 602)

#define GLUT\_HAS\_DIAL\_AND\_BUTTON\_BOX ((GLenum) 603)

#define GLUT\_HAS\_TABLET ((GLenum) 604)

#define GLUT\_NUM\_MOUSE\_BUTTONS ((GLenum) 605)

#define GLUT\_NUM\_SPACEBALL\_BUTTONS ((GLenum) 606)

#define GLUT\_NUM\_BUTTON\_BOX\_BUTTONS ((GLenum) 607)

#define GLUT\_NUM\_DIALS ((GLenum) 608)

#define GLUT\_NUM\_TABLET\_BUTTONS ((GLenum) 609)

#endif

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 13)

#define GLUT\_DEVICE\_IGNORE\_KEY\_REPEAT ((GLenum) 610)

#define GLUT\_DEVICE\_KEY\_REPEAT ((GLenum) 611)

#define GLUT\_HAS\_JOYSTICK ((GLenum) 612)

#define GLUT\_OWNS\_JOYSTICK ((GLenum) 613)

#define GLUT\_JOYSTICK\_BUTTONS ((GLenum) 614)

#define GLUT\_JOYSTICK\_AXES ((GLenum) 615)

#define GLUT\_JOYSTICK\_POLL\_RATE ((GLenum) 616)

#endif

#if (GLUT\_API\_VERSION >= 3)

/\* glutLayerGet parameters. \*/

#define GLUT\_OVERLAY\_POSSIBLE ((GLenum) 800)

#define GLUT\_LAYER\_IN\_USE ((GLenum) 801)

#define GLUT\_HAS\_OVERLAY ((GLenum) 802)

#define GLUT\_TRANSPARENT\_INDEX ((GLenum) 803)

#define GLUT\_NORMAL\_DAMAGED ((GLenum) 804)

#define GLUT\_OVERLAY\_DAMAGED ((GLenum) 805)

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

/\* glutVideoResizeGet parameters. \*/

#define GLUT\_VIDEO\_RESIZE\_POSSIBLE ((GLenum) 900)

#define GLUT\_VIDEO\_RESIZE\_IN\_USE ((GLenum) 901)

#define GLUT\_VIDEO\_RESIZE\_X\_DELTA ((GLenum) 902)

#define GLUT\_VIDEO\_RESIZE\_Y\_DELTA ((GLenum) 903)

#define GLUT\_VIDEO\_RESIZE\_WIDTH\_DELTA ((GLenum) 904)

#define GLUT\_VIDEO\_RESIZE\_HEIGHT\_DELTA ((GLenum) 905)

#define GLUT\_VIDEO\_RESIZE\_X ((GLenum) 906)

#define GLUT\_VIDEO\_RESIZE\_Y ((GLenum) 907)

#define GLUT\_VIDEO\_RESIZE\_WIDTH ((GLenum) 908)

#define GLUT\_VIDEO\_RESIZE\_HEIGHT ((GLenum) 909)

#endif

/\* glutUseLayer parameters. \*/

#define GLUT\_NORMAL ((GLenum) 0)

#define GLUT\_OVERLAY ((GLenum) 1)

/\* glutGetModifiers return mask. \*/

#define GLUT\_ACTIVE\_SHIFT 1

#define GLUT\_ACTIVE\_CTRL 2

#define GLUT\_ACTIVE\_ALT 4

/\* glutSetCursor parameters. \*/

/\* Basic arrows. \*/

#define GLUT\_CURSOR\_RIGHT\_ARROW 0

#define GLUT\_CURSOR\_LEFT\_ARROW 1

/\* Symbolic cursor shapes. \*/

#define GLUT\_CURSOR\_INFO 2

#define GLUT\_CURSOR\_DESTROY 3

#define GLUT\_CURSOR\_HELP 4

#define GLUT\_CURSOR\_CYCLE 5

#define GLUT\_CURSOR\_SPRAY 6

#define GLUT\_CURSOR\_WAIT 7

#define GLUT\_CURSOR\_TEXT 8

#define GLUT\_CURSOR\_CROSSHAIR 9

/\* Directional cursors. \*/

#define GLUT\_CURSOR\_UP\_DOWN 10

#define GLUT\_CURSOR\_LEFT\_RIGHT 11

/\* Sizing cursors. \*/

#define GLUT\_CURSOR\_TOP\_SIDE 12

#define GLUT\_CURSOR\_BOTTOM\_SIDE 13

#define GLUT\_CURSOR\_LEFT\_SIDE 14

#define GLUT\_CURSOR\_RIGHT\_SIDE 15

#define GLUT\_CURSOR\_TOP\_LEFT\_CORNER 16

#define GLUT\_CURSOR\_TOP\_RIGHT\_CORNER 17

#define GLUT\_CURSOR\_BOTTOM\_RIGHT\_CORNER 18

#define GLUT\_CURSOR\_BOTTOM\_LEFT\_CORNER 19

/\* Inherit from parent window. \*/

#define GLUT\_CURSOR\_INHERIT 100

/\* Blank cursor. \*/

#define GLUT\_CURSOR\_NONE 101

/\* Fullscreen crosshair (if available). \*/

#define GLUT\_CURSOR\_FULL\_CROSSHAIR 102

#endif

/\* GLUT initialization sub-API. \*/

GLUTAPI void APIENTRY glutInit(int \*argcp, char \*\*argv);

#if defined(\_WIN32) && !defined(GLUT\_DISABLE\_ATEXIT\_HACK)

GLUTAPI void APIENTRY \_\_glutInitWithExit(int \*argcp, char \*\*argv, void (\_\_cdecl \*exitfunc)(int));

#ifndef GLUT\_BUILDING\_LIB

static void APIENTRY glutInit\_ATEXIT\_HACK(int \*argcp, char \*\*argv) { \_\_glutInitWithExit(argcp, argv, exit); }

#define glutInit glutInit\_ATEXIT\_HACK

#endif

#endif

GLUTAPI void APIENTRY glutInitDisplayMode(unsigned int mode);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

GLUTAPI void APIENTRY glutInitDisplayString(const char \*string);

#endif

GLUTAPI void APIENTRY glutInitWindowPosition(int x, int y);

GLUTAPI void APIENTRY glutInitWindowSize(int width, int height);

GLUTAPI void APIENTRY glutMainLoop(void);

/\* GLUT window sub-API. \*/

GLUTAPI int APIENTRY glutCreateWindow(const char \*title);

#if defined(\_WIN32) && !defined(GLUT\_DISABLE\_ATEXIT\_HACK)

GLUTAPI int APIENTRY \_\_glutCreateWindowWithExit(const char \*title, void (\_\_cdecl \*exitfunc)(int));

#ifndef GLUT\_BUILDING\_LIB

static int APIENTRY glutCreateWindow\_ATEXIT\_HACK(const char \*title) { return \_\_glutCreateWindowWithExit(title, exit); }

#define glutCreateWindow glutCreateWindow\_ATEXIT\_HACK

#endif

#endif

GLUTAPI int APIENTRY glutCreateSubWindow(int win, int x, int y, int width, int height);

GLUTAPI void APIENTRY glutDestroyWindow(int win);

GLUTAPI void APIENTRY glutPostRedisplay(void);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 11)

GLUTAPI void APIENTRY glutPostWindowRedisplay(int win);

#endif

GLUTAPI void APIENTRY glutSwapBuffers(void);

GLUTAPI int APIENTRY glutGetWindow(void);

GLUTAPI void APIENTRY glutSetWindow(int win);

GLUTAPI void APIENTRY glutSetWindowTitle(const char \*title);

GLUTAPI void APIENTRY glutSetIconTitle(const char \*title);

GLUTAPI void APIENTRY glutPositionWindow(int x, int y);

GLUTAPI void APIENTRY glutReshapeWindow(int width, int height);

GLUTAPI void APIENTRY glutPopWindow(void);

GLUTAPI void APIENTRY glutPushWindow(void);

GLUTAPI void APIENTRY glutIconifyWindow(void);

GLUTAPI void APIENTRY glutShowWindow(void);

GLUTAPI void APIENTRY glutHideWindow(void);

#if (GLUT\_API\_VERSION >= 3)

GLUTAPI void APIENTRY glutFullScreen(void);

GLUTAPI void APIENTRY glutSetCursor(int cursor);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

GLUTAPI void APIENTRY glutWarpPointer(int x, int y);

#endif

/\* GLUT overlay sub-API. \*/

GLUTAPI void APIENTRY glutEstablishOverlay(void);

GLUTAPI void APIENTRY glutRemoveOverlay(void);

GLUTAPI void APIENTRY glutUseLayer(GLenum layer);

GLUTAPI void APIENTRY glutPostOverlayRedisplay(void);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 11)

GLUTAPI void APIENTRY glutPostWindowOverlayRedisplay(int win);

#endif

GLUTAPI void APIENTRY glutShowOverlay(void);

GLUTAPI void APIENTRY glutHideOverlay(void);

#endif

/\* GLUT menu sub-API. \*/

GLUTAPI int APIENTRY glutCreateMenu(void (GLUTCALLBACK \*func)(int));

#if defined(\_WIN32) && !defined(GLUT\_DISABLE\_ATEXIT\_HACK)

GLUTAPI int APIENTRY \_\_glutCreateMenuWithExit(void (GLUTCALLBACK \*func)(int), void (\_\_cdecl \*exitfunc)(int));

#ifndef GLUT\_BUILDING\_LIB

static int APIENTRY glutCreateMenu\_ATEXIT\_HACK(void (GLUTCALLBACK \*func)(int)) { return \_\_glutCreateMenuWithExit(func, exit); }

#define glutCreateMenu glutCreateMenu\_ATEXIT\_HACK

#endif

#endif

GLUTAPI void APIENTRY glutDestroyMenu(int menu);

GLUTAPI int APIENTRY glutGetMenu(void);

GLUTAPI void APIENTRY glutSetMenu(int menu);

GLUTAPI void APIENTRY glutAddMenuEntry(const char \*label, int value);

GLUTAPI void APIENTRY glutAddSubMenu(const char \*label, int submenu);

GLUTAPI void APIENTRY glutChangeToMenuEntry(int item, const char \*label, int value);

GLUTAPI void APIENTRY glutChangeToSubMenu(int item, const char \*label, int submenu);

GLUTAPI void APIENTRY glutRemoveMenuItem(int item);

GLUTAPI void APIENTRY glutAttachMenu(int button);

GLUTAPI void APIENTRY glutDetachMenu(int button);

/\* GLUT window callback sub-API. \*/

GLUTAPI void APIENTRY glutDisplayFunc(void (GLUTCALLBACK \*func)(void));

GLUTAPI void APIENTRY glutReshapeFunc(void (GLUTCALLBACK \*func)(int width, int height));

GLUTAPI void APIENTRY glutKeyboardFunc(void (GLUTCALLBACK \*func)(unsigned char key, int x, int y));

GLUTAPI void APIENTRY glutMouseFunc(void (GLUTCALLBACK \*func)(int button, int state, int x, int y));

GLUTAPI void APIENTRY glutMotionFunc(void (GLUTCALLBACK \*func)(int x, int y));

GLUTAPI void APIENTRY glutPassiveMotionFunc(void (GLUTCALLBACK \*func)(int x, int y));

GLUTAPI void APIENTRY glutEntryFunc(void (GLUTCALLBACK \*func)(int state));

GLUTAPI void APIENTRY glutVisibilityFunc(void (GLUTCALLBACK \*func)(int state));

GLUTAPI void APIENTRY glutIdleFunc(void (GLUTCALLBACK \*func)(void));

GLUTAPI void APIENTRY glutTimerFunc(unsigned int millis, void (GLUTCALLBACK \*func)(int value), int value);

GLUTAPI void APIENTRY glutMenuStateFunc(void (GLUTCALLBACK \*func)(int state));

#if (GLUT\_API\_VERSION >= 2)

GLUTAPI void APIENTRY glutSpecialFunc(void (GLUTCALLBACK \*func)(int key, int x, int y));

GLUTAPI void APIENTRY glutSpaceballMotionFunc(void (GLUTCALLBACK \*func)(int x, int y, int z));

GLUTAPI void APIENTRY glutSpaceballRotateFunc(void (GLUTCALLBACK \*func)(int x, int y, int z));

GLUTAPI void APIENTRY glutSpaceballButtonFunc(void (GLUTCALLBACK \*func)(int button, int state));

GLUTAPI void APIENTRY glutButtonBoxFunc(void (GLUTCALLBACK \*func)(int button, int state));

GLUTAPI void APIENTRY glutDialsFunc(void (GLUTCALLBACK \*func)(int dial, int value));

GLUTAPI void APIENTRY glutTabletMotionFunc(void (GLUTCALLBACK \*func)(int x, int y));

GLUTAPI void APIENTRY glutTabletButtonFunc(void (GLUTCALLBACK \*func)(int button, int state, int x, int y));

#if (GLUT\_API\_VERSION >= 3)

GLUTAPI void APIENTRY glutMenuStatusFunc(void (GLUTCALLBACK \*func)(int status, int x, int y));

GLUTAPI void APIENTRY glutOverlayDisplayFunc(void (GLUTCALLBACK \*func)(void));

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

GLUTAPI void APIENTRY glutWindowStatusFunc(void (GLUTCALLBACK \*func)(int state));

#endif

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 13)

GLUTAPI void APIENTRY glutKeyboardUpFunc(void (GLUTCALLBACK \*func)(unsigned char key, int x, int y));

GLUTAPI void APIENTRY glutSpecialUpFunc(void (GLUTCALLBACK \*func)(int key, int x, int y));

GLUTAPI void APIENTRY glutJoystickFunc(void (GLUTCALLBACK \*func)(unsigned int buttonMask, int x, int y, int z), int pollInterval);

#endif

#endif

#endif

/\* GLUT color index sub-API. \*/

GLUTAPI void APIENTRY glutSetColor(int, GLfloat red, GLfloat green, GLfloat blue);

GLUTAPI GLfloat APIENTRY glutGetColor(int ndx, int component);

GLUTAPI void APIENTRY glutCopyColormap(int win);

/\* GLUT state retrieval sub-API. \*/

GLUTAPI int APIENTRY glutGet(GLenum type);

GLUTAPI int APIENTRY glutDeviceGet(GLenum type);

#if (GLUT\_API\_VERSION >= 2)

/\* GLUT extension support sub-API \*/

GLUTAPI int APIENTRY glutExtensionSupported(const char \*name);

#endif

#if (GLUT\_API\_VERSION >= 3)

GLUTAPI int APIENTRY glutGetModifiers(void);

GLUTAPI int APIENTRY glutLayerGet(GLenum type);

#endif

/\* GLUT font sub-API \*/

GLUTAPI void APIENTRY glutBitmapCharacter(void \*font, int character);

GLUTAPI int APIENTRY glutBitmapWidth(void \*font, int character);

GLUTAPI void APIENTRY glutStrokeCharacter(void \*font, int character);

GLUTAPI int APIENTRY glutStrokeWidth(void \*font, int character);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

GLUTAPI int APIENTRY glutBitmapLength(void \*font, const unsigned char \*string);

GLUTAPI int APIENTRY glutStrokeLength(void \*font, const unsigned char \*string);

#endif

/\* GLUT pre-built models sub-API \*/

GLUTAPI void APIENTRY glutWireSphere(GLdouble radius, GLint slices, GLint stacks);

GLUTAPI void APIENTRY glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);

GLUTAPI void APIENTRY glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);

GLUTAPI void APIENTRY glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);

GLUTAPI void APIENTRY glutWireCube(GLdouble size);

GLUTAPI void APIENTRY glutSolidCube(GLdouble size);

GLUTAPI void APIENTRY glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);

GLUTAPI void APIENTRY glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);

GLUTAPI void APIENTRY glutWireDodecahedron(void);

GLUTAPI void APIENTRY glutSolidDodecahedron(void);

GLUTAPI void APIENTRY glutWireTeapot(GLdouble size);

GLUTAPI void APIENTRY glutSolidTeapot(GLdouble size);

GLUTAPI void APIENTRY glutWireOctahedron(void);

GLUTAPI void APIENTRY glutSolidOctahedron(void);

GLUTAPI void APIENTRY glutWireTetrahedron(void);

GLUTAPI void APIENTRY glutSolidTetrahedron(void);

GLUTAPI void APIENTRY glutWireIcosahedron(void);

GLUTAPI void APIENTRY glutSolidIcosahedron(void);

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 9)

/\* GLUT video resize sub-API. \*/

GLUTAPI int APIENTRY glutVideoResizeGet(GLenum param);

GLUTAPI void APIENTRY glutSetupVideoResizing(void);

GLUTAPI void APIENTRY glutStopVideoResizing(void);

GLUTAPI void APIENTRY glutVideoResize(int x, int y, int width, int height);

GLUTAPI void APIENTRY glutVideoPan(int x, int y, int width, int height);

/\* GLUT debugging sub-API. \*/

GLUTAPI void APIENTRY glutReportErrors(void);

#endif

#if (GLUT\_API\_VERSION >= 4 || GLUT\_XLIB\_IMPLEMENTATION >= 13)

/\* GLUT device control sub-API. \*/

/\* glutSetKeyRepeat modes. \*/

#define GLUT\_KEY\_REPEAT\_OFF 0

#define GLUT\_KEY\_REPEAT\_ON 1

#define GLUT\_KEY\_REPEAT\_DEFAULT 2

/\* Joystick button masks. \*/

#define GLUT\_JOYSTICK\_BUTTON\_A 1

#define GLUT\_JOYSTICK\_BUTTON\_B 2

#define GLUT\_JOYSTICK\_BUTTON\_C 4

#define GLUT\_JOYSTICK\_BUTTON\_D 8

GLUTAPI void APIENTRY glutIgnoreKeyRepeat(int ignore);

GLUTAPI void APIENTRY glutSetKeyRepeat(int repeatMode);

GLUTAPI void APIENTRY glutForceJoystickFunc(void);

/\* GLUT game mode sub-API. \*/

/\* glutGameModeGet. \*/

#define GLUT\_GAME\_MODE\_ACTIVE ((GLenum) 0)

#define GLUT\_GAME\_MODE\_POSSIBLE ((GLenum) 1)

#define GLUT\_GAME\_MODE\_WIDTH ((GLenum) 2)

#define GLUT\_GAME\_MODE\_HEIGHT ((GLenum) 3)

#define GLUT\_GAME\_MODE\_PIXEL\_DEPTH ((GLenum) 4)

#define GLUT\_GAME\_MODE\_REFRESH\_RATE ((GLenum) 5)

#define GLUT\_GAME\_MODE\_DISPLAY\_CHANGED ((GLenum) 6)

GLUTAPI void APIENTRY glutGameModeString(const char \*string);

GLUTAPI int APIENTRY glutEnterGameMode(void);

GLUTAPI void APIENTRY glutLeaveGameMode(void);

GLUTAPI int APIENTRY glutGameModeGet(GLenum mode);

#endif

#ifdef \_\_cplusplus

}

#endif

#ifdef GLUT\_APIENTRY\_DEFINED

# undef GLUT\_APIENTRY\_DEFINED

# undef APIENTRY

#endif

#ifdef GLUT\_WINGDIAPI\_DEFINED

# undef GLUT\_WINGDIAPI\_DEFINED

# undef WINGDIAPI

#endif

#ifdef GLUT\_DEFINED\_\_\_CDECL

# undef GLUT\_DEFINED\_\_\_CDECL

# undef \_\_cdecl

#endif

#ifdef GLUT\_DEFINED\_\_CRTIMP

# undef GLUT\_DEFINED\_\_CRTIMP

# undef \_CRTIMP

#endif

#endif /\* \_\_glut\_h\_\_ \*/

**main.cpp :**

#include "CMyOpenGLInit.h"

void main(int argc, char\* argv[])

{

CMyOpenGLInit MyGame(argc, argv);

}