Бруцкая Анастасия 251001

Лаборатораня работа №1

#define ID\_ROTATE\_LEFT 5001  
#define ID\_ROTATE\_RIGHT 5002  
  
ACCEL accelTable[] = {  
 {FSHIFT | FVIRTKEY, VK\_LEFT, ID\_ROTATE\_LEFT},  
 {FSHIFT | FVIRTKEY, VK\_RIGHT, ID\_ROTATE\_RIGHT}  
};

void RotatePoints(float &x, float &y, int centerX, int centerY, int angle) {  
 float angleRadians = angle \* static\_cast<float>(M\_PI) / 180.0f;  
  
 float dx = x - centerX;  
 float dy = y - centerY;  
  
 float angleCos = cos(angleRadians);  
 float angleSin = sin(angleRadians);  
  
 float newX = dx \* angleCos - dy \* angleSin;  
 float newY = dx \* angleSin + dy \* angleCos;  
  
 x = newX + centerX;  
 y = newY + centerY;  
}  
  
double TriangleArea(POINT A, POINT B, POINT C) {  
 return 0.5 \* abs(A.x \* (B.y - C.y) + B.x \* (C.y - A.y) + C.x \* (A.y - B.y));  
}  
  
bool IsPointInTriangle(POINT P, POINT A, POINT B, POINT C) {  
 double areaABC = TriangleArea(A, B, C);  
  
 double areaPAB = TriangleArea(P, A, B);  
 double areaPBC = TriangleArea(P, B, C);  
 double areaPCA = TriangleArea(P, C, A);  
  
 return abs(areaABC - (areaPAB + areaPBC + areaPCA)) < 1e-5;  
}  
  
void calculateTrianglePoints(POINT points[3], int centerX, int centerY, double angle) {  
  
 points[0] = {0, (LONG) (-IMAGE\_HEIGHT / 2)};  
 points[1] = {(LONG) (-IMAGE\_WIDTH / 2), (LONG) (IMAGE\_HEIGHT / 2)};  
 points[2] = {(LONG) (IMAGE\_WIDTH / 2), (LONG) (IMAGE\_HEIGHT / 2)};  
  
 float angleRadians = angle \* static\_cast<float>(M\_PI) / 180.0f;  
  
 for (int i = 0; i < 3; ++i) {  
 double x = points[i].x \* cos(angleRadians) - points[i].y \* sin(angleRadians);  
 double y = points[i].x \* sin(angleRadians) + points[i].y \* cos(angleRadians);  
 points[i].x = (LONG) (centerX + x);  
 points[i].y = (LONG) (centerY + y);  
 }  
}  
  
void CheckTriangleBounds(HWND hWnd) {  
 RECT clientRect;  
 GetClientRect(hWnd, &clientRect);  
  
 POINT points[3];  
 calculateTrianglePoints(points, imageCenterX, imageCenterY, angle);  
  
 for (int i = 0; i < 3; ++i) {  
 if (points[i].x < 0) imageCenterX += MOVE\_SPEED;  
 if (points[i].x > clientRect.right) imageCenterX -= MOVE\_SPEED;  
 if (points[i].y < 0) imageCenterY += MOVE\_SPEED;  
 if (points[i].y > clientRect.bottom) imageCenterY -= MOVE\_SPEED;  
 }  
}  
  
  
bool CheckMouseCoordinatesForSquare(float mouseX, float mouseY) {  
 RotatePoints(mouseX, mouseY, imageCenterX, imageCenterY, -angle);  
  
 return (mouseX >= imageCenterX - IMAGE\_WIDTH / 2 && mouseX <= imageCenterX + IMAGE\_WIDTH / 2 &&  
 mouseY >= imageCenterY - IMAGE\_HEIGHT / 2 && mouseY <= imageCenterY + IMAGE\_HEIGHT / 2);  
  
}  
  
void CheckRotatedSquareBounds(HWND hWnd) {  
 RECT clientRect;  
 GetClientRect(hWnd, &clientRect);  
  
 float topLeftX = imageCenterX - IMAGE\_WIDTH / 2;  
 float topLeftY = imageCenterY - IMAGE\_HEIGHT / 2;  
  
 float topRightX = imageCenterX + IMAGE\_WIDTH / 2;  
 float topRightY = imageCenterY - IMAGE\_HEIGHT / 2;  
  
 float bottomLeftX = imageCenterX - IMAGE\_WIDTH / 2;  
 float bottomLeftY = imageCenterY + IMAGE\_HEIGHT / 2;  
  
 float bottomRightX = imageCenterX + IMAGE\_WIDTH / 2;  
 float bottomRightY = imageCenterY + IMAGE\_HEIGHT / 2;  
  
 RotatePoints(topLeftX, topLeftY, imageCenterX, imageCenterY, angle);  
 RotatePoints(topRightX, topRightY, imageCenterX, imageCenterY, angle);  
 RotatePoints(bottomLeftX, bottomLeftY, imageCenterX, imageCenterY, angle);  
 RotatePoints(bottomRightX, bottomRightY, imageCenterX, imageCenterY, angle);  
  
 float minX = findMinElement(new float[]{topLeftX, topRightX, bottomLeftX, bottomRightX}, 4);  
 float minY = findMinElement(new float[]{topLeftY, topRightY, bottomLeftY, bottomRightY}, 4);  
 float maxX = findMaxElement(new float[]{topLeftX, topRightX, bottomLeftX, bottomRightX}, 4);  
 float maxY = findMaxElement(new float[]{topLeftY, topRightY, bottomLeftY, bottomRightY}, 4);  
  
 if (minX < 0) {  
 imageCenterX += -minX;  
 }  
 if (maxX > clientRect.right) {  
 imageCenterX -= (maxX - clientRect.right);  
 }  
 if (minY < 0) {  
 imageCenterY += -minY;  
 }  
 if (maxY > clientRect.bottom) {  
 imageCenterY -= (maxY - clientRect.bottom);  
 }  
  
}  
  
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,  
 WPARAM wParam, LPARAM lParam) {  
 switch (message) {  
 case WM\_CREATE: {  
 normalImage = LoadImageFromFile(PATH\_TO\_CLOSE\_MOUTH\_CAT\_IMAGE);  
 clickedImage = LoadImageFromFile(PATH\_TO\_OPEN\_MOUTH\_CAT\_IMAGE);  
  
 if (!normalImage || normalImage->GetLastStatus() != *Ok* || !clickedImage ||  
 clickedImage->GetLastStatus() != *Ok*) {  
 MessageBox(hWnd, "Не удалось загрузить изображения!", "Ошибка", MB\_OK | MB\_ICONERROR);  
 PostQuitMessage(0);  
 }  
 break;  
 }  
 case WM\_PAINT: {  
 PAINTSTRUCT ps;  
 HDC hdc = BeginPaint(hWnd, &ps);  
  
 HDC memDC = CreateCompatibleDC(hdc);  
 HBITMAP memBitmap = CreateCompatibleBitmap(hdc, ps.rcPaint.right - ps.rcPaint.left,  
 ps.rcPaint.bottom - ps.rcPaint.top);  
 SelectObject(memDC, memBitmap);  
  
 Graphics graphics(memDC);  
  
 HBRUSH hBrush = CreateSolidBrush(RGB(255, 255, 255));  
 FillRect(memDC, &ps.rcPaint, hBrush);  
 DeleteObject(hBrush);  
  
 graphics.TranslateTransform(imageCenterX, imageCenterY);  
 graphics.RotateTransform(angle);  
 graphics.TranslateTransform(-imageCenterX, -imageCenterY);  
  
 if (isImageChanged && clickedImage) {  
 graphics.DrawImage(clickedImage, imageCenterX - IMAGE\_WIDTH / 2, imageCenterY - IMAGE\_HEIGHT / 2,  
 IMAGE\_WIDTH, IMAGE\_HEIGHT);  
 } else if (normalImage) {  
 graphics.DrawImage(normalImage, imageCenterX - IMAGE\_WIDTH / 2, imageCenterY - IMAGE\_HEIGHT / 2,  
 IMAGE\_WIDTH, IMAGE\_HEIGHT);  
 }  
  
  
 BitBlt(hdc, ps.rcPaint.left, ps.rcPaint.top, ps.rcPaint.right - ps.rcPaint.left,  
 ps.rcPaint.bottom - ps.rcPaint.top, memDC, 0, 0, SRCCOPY);  
  
 DeleteObject(memBitmap);  
 DeleteDC(memDC);  
  
 EndPaint(hWnd, &ps);  
 break;  
 }  
 case WM\_ERASEBKGND:  
 return 1;  
 case WM\_LBUTTONDOWN: {  
 POINT clickPoint;  
 clickPoint.x = LOWORD(lParam);  
 clickPoint.y = HIWORD(lParam);  
  
 int mouseX = LOWORD(lParam);  
 int mouseY = HIWORD(lParam);  
  
 POINT points[3];  
 calculateTrianglePoints(points, imageCenterX, imageCenterY, angle);  
  
 if (isTriangleImage) {  
  
 if (IsPointInTriangle(clickPoint, points[0], points[1], points[2])) {  
 isImageChanged = true;  
 isImageDragging = true;  
 };  
 } else {  
 if (CheckMouseCoordinatesForSquare(mouseX, mouseY)) {  
 isImageChanged = true;  
 isImageDragging = true;  
 }  
 }  
 UpdateSpritePosition(hWnd);  
 break;  
 }  
 case WM\_LBUTTONUP: {  
 isImageChanged = false;  
 isImageDragging = false;  
 UpdateSpritePosition(hWnd);  
 break;  
 }  
 case WM\_MOUSEMOVE: {  
 if (isImageDragging) {  
 imageCenterX = LOWORD(lParam) - DRAGGING\_OFFSET;  
 imageCenterY = HIWORD(lParam) - DRAGGING\_OFFSET;  
  
 if (isTriangleImage) {  
 CheckTriangleBounds(hWnd);  
 } else {  
 CheckRotatedSquareBounds(hWnd);  
 }  
 UpdateSpritePosition(hWnd);  
 }  
 break;  
 }  
 case WM\_KEYDOWN: {  
  
 if (!IsEnglishKeyboardLayout()) {  
 break;  
 }  
  
 switch (wParam) {  
 case 'T':  
 normalImage = LoadImageFromFile(PATH\_TO\_TRIANGLE\_IMAGE);  
 isTriangleImage = true;  
 break;  
 case 'C':  
 normalImage = LoadImageFromFile(PATH\_TO\_CLOSE\_MOUTH\_CAT\_IMAGE);  
 isTriangleImage = false;  
 break;  
 case 'W':  
 case VK\_UP:  
 imageCenterY -= MOVE\_SPEED;  
 break;  
 case 'S':  
 case VK\_DOWN:  
 imageCenterY += MOVE\_SPEED;  
 break;  
 case 'A':  
 case VK\_LEFT:  
 imageCenterX -= MOVE\_SPEED;  
 break;  
 case 'D':  
 case VK\_RIGHT:  
 imageCenterX += MOVE\_SPEED;  
 break;  
 case VK\_ESCAPE: {  
 int result = MessageBox(hWnd, "Are you sure you want to exit", "Exit Confirmation",  
 MB\_YESNO | MB\_ICONQUESTION);  
 if (result == IDYES) {  
 PostQuitMessage(0);  
 }  
 break;  
 }  
 }  
  
 if (isTriangleImage) {  
 CheckTriangleBounds(hWnd);  
 } else {  
 CheckRotatedSquareBounds(hWnd);  
 }  
 UpdateSpritePosition(hWnd);  
 break;  
 }  
 case WM\_MOUSEWHEEL: {  
 int zDelta = GET\_WHEEL\_DELTA\_WPARAM(wParam);  
  
 if (GetKeyState(VK\_SHIFT) & 0x8000) {  
 imageCenterX += (zDelta > 0 ? -MOVE\_SPEED : MOVE\_SPEED);  
 } else {  
 imageCenterY += (zDelta > 0 ? -MOVE\_SPEED : MOVE\_SPEED);  
 }  
  
 if (isTriangleImage) {  
 CheckTriangleBounds(hWnd);  
 } else {  
 CheckRotatedSquareBounds(hWnd);  
 }  
 UpdateSpritePosition(hWnd);  
 break;  
 }  
 case WM\_COMMAND: {  
 switch (LOWORD(wParam)) {  
 case ID\_ROTATE\_LEFT:  
 angle -= 5;  
 if (angle < 0) angle += 360;  
 break;  
 case ID\_ROTATE\_RIGHT:  
 angle += 5;  
 if (angle >= 360) angle -= 360;  
 break;  
 }  
  
 UpdateSpritePosition(hWnd);  
 break;  
 }  
 case WM\_DESTROY: {  
  
 if (normalImage) {  
 delete normalImage;  
 }  
 if (clickedImage) {  
 delete clickedImage;  
 }  
  
 PostQuitMessage(0);  
 break;  
 }  
 default:  
 return DefWindowProc(hWnd, message, wParam, lParam);  
 }  
 return 0;  
}