Numerical solution of anisotropic heat equation in curved domain

Oleh Krehel
Erasmus Mundus Mathmods
University of Nice-Sophia Antipolis

July 20, 2009

Abstract

This document concentrates on *practical implementation* of the Finite Element Method in C++ for the heat equation, without resorting to *blackbox* libraries on all steps, from GUI modelling of the domain using second order Bezier splines, downto solving the discretized problem with a sparse direct solver.

0.1 The Delaunay algorithm for domain triangulation

This method is popular because of its property of maximizing the minimal angle among all possible triangulation of a set of vertices. This is really useful for the FEM, because small angles present in trianguation result in larger errors and lower convergence, expecially for low orders of approximation.

My program uses Jonathan Shewchuk's *Triangle* package, mainly because it's very robust and fast.

0.1.1 Basic definitions

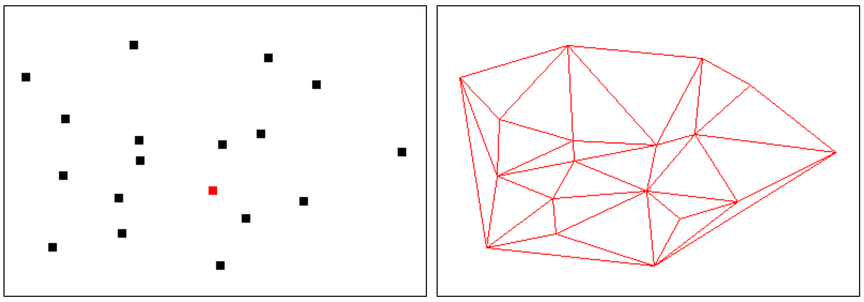
Definition 1 *For a given set of points V, we call a* triangulation *a set of triangles T, such that:*

- *all points of T constitute V*
- *none of the triangles in T overlap*
- *the union of all triangles of T is the convex hull of V*
- *T crosses V only in nodes.*

Definition 2 *A* Delaunay triangulation *D of a vertex set V is a graph with the following property: if $u, v \in V$, then $uv \in D \Leftrightarrow \exists$ circle, that passes through u, v that doesn't contain inside any point from V (this is called the Delaunay property of the edge).*

From this definition it's obvious that for a given set of points, the Delaunay triangulation is unique, as we can determine for each edge if it's present in the triangulation.

On the pictures below, we have a set of vertices and its Delaunay triangulation.



Definition 3 *A triangle is* Delaunay *if its circumcircle doesn't have any points inside.*

0.2 Flipping Algorithm

The Flipping Algorithm starts from any triangualtion of set of points V, and looks for an edge that isn't Delaunay. On the picture on the left, the edge has the local Delaunay property, on the right – no. When it finds it, the edge is removed, creating a quad, and then inserted back as the other diagonal. Not all edges is triangualtion may be flipped – the intermediate quad is not always convex.

Definition 4 *An edge* e *of triangulation is locally Delaunay if it has the Delaunay property with respect to just to the vertices of two triangles that contain* e*. Then for a triangle that has the Delaunay property, we can say that it's globally Delaunay.*

Lemma 1 *Let* e *be an edge in triangulation. Then either* e *is locally Delaunay, or it can be flipped, and the resulting edge will be locally Delaunay.*

Lemma 2 *Let T be a triangualtion, and all its edges are locally Delaunay. Then all of its edges are globally Delaunay.*

Lemma 3 *For a set of n points, the flipping algorithm terminates after $O(n^2)$ flips, creating a Delaunay triangualtion.*

Theorem 1 *Let V be a set of points on a plane. If none of four points lie on the same circle, then the Delaunay triangualtion is in fact a triangulation and can be obtained by the flipping algorithm.*

0.3 PSLG and the Constrained Delaunay triangulation

Note that the Delaunay triangulation is not exactly what we need for our finite element mesh. Usually on the input we have not the points, but the boundary of domain, and on the output we expect not only triangulation with maximized minimum angle, but with some additional properties(for instance, we usually scpecify the maximum area of a triangle). Also the Delaunay triangulation gives us a convex hull of a set of points, which is not what we want if the domain we are modelling is not convex.

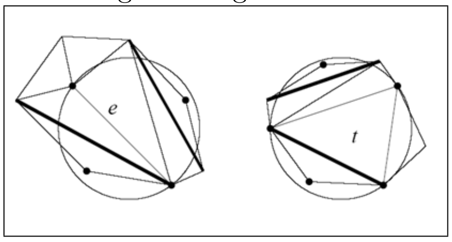
This is why most theoretical triangulation methods take a PSLG as an input.

Definition 5 *A* Planar Straight Line Graph(PSLG) *is a collection of vertices and segments. Segments are edges whose endpoints are vertices in the PSLG, and whose presence in any mesh generated from the PSLG is enforced. Segments of PSLG do not cross, except in endpoints.*

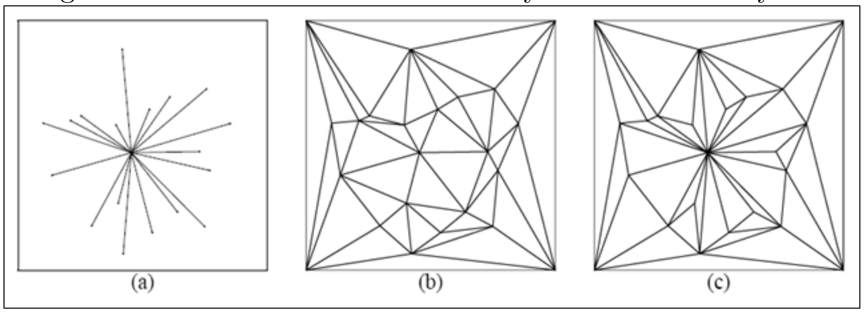
Definition 6 *A* constrained Delaunay triangulation *of a PSLG is similar to a Delaunay triangulation, but each PSLG segment is present as a single edge in the triangulation. A constrained Delaunay triangulation is not truly a Delaunay triangulation. Some of its triangles might not be Delaunay, but they are all constrained Delaunay.*

Definition 7 *An edge or a triangle is constrained Delaunay if it doesn't cross/cover a segment of PSLG(except when an edge belongs to PSLG), and its circumcircle doesn't cover any point of PSLG that is visible from the middle of the triangle or the edge(the line joining the middle point and a point of PSLG doesn't cross a segment of PSLG).*

On the picture below the edge e and the triangle t are constrained Delaunay. Although their circumcircles cover vertices of PSLG, they aren't visible through the segments.



On the next picture there's (a)–PSLG,(b)–DT,(c)–CDT. As we can see, some edges of CDT are constrained Delaunay but not Delaunay.



If no four vertices of PSLG lie on a same circle, then constrianed Delaunay triangulation is a set of constrianed Delaunay triangles, it contains all segments of PSLG.

CDT can be built by flipping algorithm, starting from a triangulation that contains all segments of PSLG:

- start from any triangulation of the vertices of PSLG(for instance, Delaunay)
- one by one, we check if it contains all segments
- if it's not there, then:
  - remove from triangualtion all edges that intersect it
  - insert the segment
  - triangulate the resulting two polygons(that form on either side of the segment)

After we've found the triangulation that contains all the segments of PSLG, we can apply the flipping algorithm to it(with restriction that segments cannot be flipped).

Theorem 2 *Let X be PSLG with three of more points, that aren't colinear, and no four points lie on the same cirlce. Then CDT is a triangualtion and it can be obtained by flipping algorithm.*

Theorem 3 *Of all the constrained triangulations, CDT maximizes the minimal angle and minimizes the maximal circumcircle.*

Definition 8 *A* conforming Delaunay triangulation(CDT) *of a PSLG is a true Delaunay triangulation in which each PSLG segment may have been subdivided into several edges by insertion of additional vertices, called* Steiner points*. Steiner points are necessary to allow the segments to exist in the mesh while maintaining the Delaunay property. Steiner points are also inserted to meet constraints on the miniumum angle and maximum triangle area.*

The latter is a triangulation that a Finite Element method can work with.

0.4 Domain generation, based on a list of points and normals

Though it's possible to construct any domain that the program can handle within the program's editor, sometimes for purposes of research or automation we need to generate a domain based on a list of points and normals in each point. This way it's possible to take a parametrizable domain like an ellipse, quickly generate these lists with as many sample points as we wish and then construct the domain with them.

The program uses the second order Bezier splines to approximate the curved segments:

$$s(t) = t^2 p_1 + 2t(1-t)m + (1-t)^2 p_2 \tag{1}$$

where p_1 and p_2 are the endpoints and m is the control point.

So for each curved segment we need to have the coordinates of p_1, p_2 and m. But instead we have the coordinates of p_1 , p_2 and the normals n_1 and n_2. Actually, this is even a little more information than we need, it's more suitalbe for third order Bezier spline. We can still use it for the second order Bezier spline, but a bit of this informaiton will be lost. To put it other way, we have 4 degrees of freedom on the input, but second order Bezier spline only uses 3. So the task is to find the coordinates of m based on coordinates of endpoints and normals.

When we derivate $s(t)$ and evaluate

$$\begin{aligned} s'(t)_{|t=0} &= 2(m - p_2) \\ s'(t)_{|t=1} &= 2(p_1 - m) \end{aligned} \tag{2}$$

it's easy to see that m lies on crossing of tangents τ_1 and τ_2 in p_1 and p_2. We can find these tangents up to a constant multiplier from

$$\begin{aligned} \tau_1 \perp n_1 \quad &\Leftrightarrow \quad (\tau_1, n_1) = 0 \\ \tau_2 \perp n_2 \quad &\Leftrightarrow \quad (\tau_2, n_2) = 0 \end{aligned} \tag{3}$$

Then the coordinates of m can be found from intersection of these two lines:

$$\begin{cases} (x - x_1, y - y_1) \cdot n_1 &= 0 \\ (x - x_2, y - y_2) \cdot n_2 &= 0 \end{cases} \tag{4}$$

The solution of this system is:

$$\begin{aligned} x &= \frac{x_1 n_{1x} n_{2y} - x_2 n_{2x} n_{1y} + n_{1y} n_{2y}(y_1 - y_2)}{n_{1x} n_{2y} - n_{2x} n_{1y}} \\ y &= \frac{y_1 n_{1y} n_{2x} - y_2 n_{2y} n_{1x} + n_{1x} n_{2y}(x_1 - y_2)}{n_{1y} n_{2x} - n_{2y} n_{1x}} \end{aligned} \tag{5}$$

Now using this method of input we can create an ellipse domain:

$$\begin{aligned} x(t) &= x_0 + acos(t) \\ y(t) &= y_0 + bsin(t) \quad t \in [0, 2\pi] \end{aligned} \tag{6}$$

Just distribute t_i uniformely on $[0, 2\pi]$, and evaluate x_i and y_i. To evaluate n_i, we can derivate once to get the tangent and then get the normal from (3).

$$\begin{aligned} x_i &= x_0 + acos(t_i) \\ y_i &= y_0 + bsin(t_i) \\ n_i &= (bsin(t_i), acos(t_i)) \end{aligned} \tag{7}$$

0.5 Computing matrices on reference triangle

To compute the elements

$$\int_K \nabla N_i^K \cdot k(x,y) \cdot \nabla N_j^K and \int_K N_i^K N_j^K \tag{8}$$

we need an effective way of evaluating the functions N_i^K and their gradients. This can be done by moving to the so-called reference element.

For triangles, the reference element is the triangle with vertices

$$\hat{p_1} = (0,0), \quad \hat{p_1} = (1,0) \quad \hat{p_1} = (0,1) \tag{9}$$

The local nodal functions in the reference triangle for $\mathbf{P}_1$ are:

$$\hat{N}_1 = 1 - \xi - \eta, \quad \hat{N}_2 = \xi, \quad \hat{N}_3 = \eta \tag{10}$$

Let us now take the three vertices of a triangle K

$$p_1^K = (x_1, y_1), \quad p_2^K = (x_2, y_2), \quad p_3^K = (x_3, y_3) \tag{11}$$

The following transformation

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{12}$$

maps the triangle $\hat{K}$ bijectively into K. Call this transformation F_K.

$$F_K(\hat{p}_i) = p_i^K, \quad i = 1, 2, 3 \tag{13}$$

It is simple now to prove that

$$\hat{N}_i = N_i^K \circ F_K, \quad i = 1, 2, 3 \tag{14}$$

or, what is the same

$$N_i^K = \hat{N}_i \circ F_K^{-1} \quad i = 1, 2, 3 \tag{15}$$

In the last expression, what we have is

$$N_i^K(x,y) = \hat{N}_i(F_K^{-1}(x,y)) \tag{16}$$

Since computing F_K^{-1} is straightforward from the explicit expression for F_K, this formula gives a simple way of evaluating the functions N_i^K.

To evaluate the gradient of N_i^K we have to apply the chain rule:

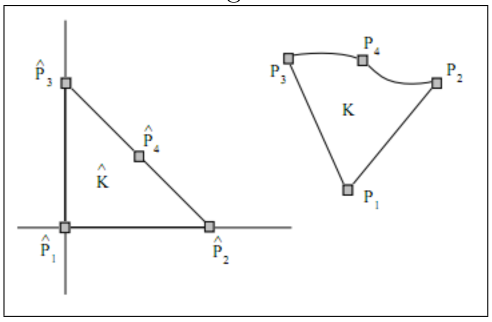
$$B_K^T(\nabla\phi \circ F_K) = \hat{\nabla}(\phi \circ F_K) \tag{17}$$

B_K^T is the transposed of the matrix of the linear transformation F_K (12). Taking $\phi = N_i^K$ in this expression, we obtain:

$$\nabla N_i^K = B_K^{-T}\big((\hat{\nabla}\hat{N}_i) \circ F_K^{-1}\big) \tag{18}$$

0.6 Isoparametric elements

The first issue when dealing with curved domains, is that *all boundary nodes of the triangulation must be placed on the real boundary.* This means that is we want a smaller grid, we cannot simply subdivide the old one, we have to call back the triangualtor.



On the picture above, we have the reference triangle and a deformation of the image triangle.

Let us now call p_4^K to the midpoint of the segment that joins $\hat{p_2}$ and $\hat{p_3}$, that is $\hat{p_4} = (\frac{1}{2}, \frac{1}{2})$.

Take a fourth point in the physical space, $p_4^K = (x_4, y_4)$ and compute its deviation from the midpoint of p_2^K and p_3^K

$$\begin{bmatrix}\delta_x \\ \delta_y\end{bmatrix} = \begin{bmatrix}x_4 \\ y_4\end{bmatrix} - \begin{bmatrix}\frac{x_2+x_3}{2} \\ \frac{y_2+y_3}{2}\end{bmatrix} \tag{19}$$

Finally take the transformation $F_K : \hat{K} \to \mathbf{R}^2$ given by

$$F_K(\xi, \eta) = F_K^0(\xi, \eta) + 4\xi\eta \begin{bmatrix}\delta_x \\ \delta_y\end{bmatrix} \tag{20}$$

Note that this is a linear transformation F_K^0 plus a correction term. The transformation F_K satisfies the following properties:

- It sends the chosen points in the reference domain to the ones in the physical space
$$F_K(\hat{p}_i) = p_i^K, \quad i = 1, \dots, 4 \tag{21}$$

- If $\xi = 0$, then
$$F_K(0, t) = F_K^0(0, t) \tag{22}$$
This means that the image of the vertical edge in reference coordinates is the segment joining p_1^K and p_3^K, covered at constant velocity, as if we were using the linear transformation. The same thing happens to the horizontal side of$\hat{K}$.

- If p_4^K is aligned with p_2^K and p_3^K, then the image of the edge that joins $\hat{p}_2$ and $\hat{p}_3$ is the segment that joins p_2^K and p_3^K. However, this segment is parameterized at constant velocity only when p_4^K is the midpoint of p_2^K and p_3^K(in that case $\delta_x = \delta_y = 0$ and we have only the linear term in the transformation F_K).

- The Jacobian matrix of F_K is not constant:

$$B_K = DF(\xi, \eta) = B_K^0 + 4 \begin{bmatrix} \eta \\ \xi \end{bmatrix} \begin{bmatrix} \delta_x & \delta_y \end{bmatrix} = \begin{bmatrix} x_2 - x_1 + 4\eta\delta_x & x_3 - x_1 + 4\eta\delta_y \\ y_2 - y_1 + 4\xi\delta_x & y_3 - y_1 + 4\xi\delta_y \end{bmatrix} \tag{23}$$

When p_4^K is not too far from the midpoint of p_2^K and p_3^K, that is, when the deviation (δ_x, δ_y) is not too large, it is possible to prove that the image if $\hat{K}$ under this transformation $K = F_K(\hat{K})$ is mapped bijectively from the reference element and therefore we can construct an inverse to $F_K : \hat{K} \to K$.

0.6.1 Computing the local integrals

Let's begin with the mass matrix, which will be used to sovle the time-dependant problem.

We have a formula for the local basis functions

$$N_i^K = \hat{N}_i \circ F_K^{-1} \tag{24}$$

Instead of integrating on K which would require us to invert F_K, we move everything to the reference domain:

$$\int_K N_i^K N_j^K = \int_{\hat{K}} |detB_K| \hat{N}_i \hat{N}_j \tag{25}$$

With this strategy, the integral is defined on a plain triangle and we just need to compute the non-constant determinant of

$$B_K = \begin{bmatrix} x_2 - x_1 + 4\eta\delta_x & x_3 - x_1 + 4\eta\delta_y \\ y_2 - y_1 + 4\xi\delta_x & y_3 - y_1 + 4\xi\delta_y \end{bmatrix} \tag{26}$$

on the chosen quadrature points.

We integrate the stiffness matrix on the reference triangle for the same reasons as with the mass matrix. The integral is:

$$\int_{\hat{K}} |detB_K| ((B_K^{-T} \nabla \hat{N}_i)^T \cdot k(x(\xi, \eta), y(\xi, \eta)) \cdot (B_K^{-T} \nabla \hat{N}_i) \tag{27}$$

0.7 Formulation of integrals over a triangular area

The Finite Element method for 2D problems with triangular elements requiers the numerical integration of shape functions on a triangle. Since an affine transformation makes it possible to transform any triangle into a standard triangle T with coordinates {(0,0), (0,1), (1,0)}, we have to consider just the numerical integration on T. The integral of an arbitrary function f over the surface of a triangle T is given by:

$$I = \iint_T f(x,y)\,\mathrm{d}x\,\mathrm{d}y = \int_0^1 dx \int_0^{1-x} f(x,y)\,\mathrm{d}y = \int_0^1 dy \int_0^{1-y} f(x,y)\,\mathrm{d}x \tag{28}$$

Now we have to find the value of the integral by a quadrature formula:

$$I = \sum_{m=1}^{N} c_m f(x_m, y_m) \tag{29}$$

where c_m are the weights associated with specifice points (x_m, y_m) and N is the number of pivotal points related to the required precision.

The integral I of equation (28) can be transformed into an integral over the surface of the square: $\{(u,v)| \quad 0 \leq u, v \leq 1\}$, by substitution:

$$x = u, y = (1-u)v \tag{30}$$

Then the determinant of the Jacobian and the differential area are:

$$\frac{\partial(x,y)}{\partial(u,v)} = \frac{\partial x}{\partial u}\frac{\partial y}{\partial v} - \frac{\partial x}{\partial v}\frac{\partial y}{\partial u} = (1)(1-u) - 0(-v) = 1-u$$

and

$$\mathrm{d}x\,\mathrm{d}y = \frac{\partial(x,y)}{\partial(u,v)}\,\mathrm{d}u\,\mathrm{d}v = (1-u)\,\mathrm{d}u\,\mathrm{d}v \tag{31}$$

Then on using equations (30) and (31) in equation (28), we have

$$\int_0^1 \int_0^{1-x} f(x,y)\,\mathrm{d}x\,\mathrm{d}y = \int_0^1 \int_0^1 f(u,(1-u)v)(1-u)\,\mathrm{d}u\,\mathrm{d}v \tag{32}$$

The integral I of equation (32) can be transformed further into an integral over the standard 2-square: $\{(\xi,\eta)| -1 \leq \xi, \eta \leq 1\}$ by the substitution

$$u = (1+\xi)/2, v = (1+\eta)/2, \tag{33}$$

then clearly the determinant of the Jacobian and the differential area are:

$$\frac{\partial(u,v)}{\partial(\xi,\eta)} = \frac{\partial u}{\partial \xi}\frac{\partial v}{\partial \eta} - \frac{\partial u}{\partial \eta}\frac{\partial v}{\partial \xi} = (1/2)(1/2) - (0)(0) = 1/4$$

$$\mathrm{d}u\,\mathrm{d}v = \frac{\partial(u,v)}{\partial(\xi,\eta)}\,\mathrm{d}\xi\,\mathrm{d}\eta = \frac{1}{4}\,\mathrm{d}\xi\,\mathrm{d}\eta \tag{34}$$

Now, on using equations (33) and (34) in equation (32), we have:

$$\begin{aligned} I &= \int_0^1 \int_0^{1-x} f(x,y)\,\mathrm{d}y\,\mathrm{d}x = \int_0^1 \int_0^1 f(u,(1-u)v)(1-u)\,\mathrm{d}u\,\mathrm{d}v \\ &= \int_{-1}^1 \int_{-1}^1 f(\frac{1+\xi}{2}, \frac{(1-\xi)(1+\eta)}{4})(\frac{1-\xi}{8})\,\mathrm{d}\xi\,\mathrm{d}\eta \end{aligned} \tag{35}$$

Equation (35) represents an integral over the surface of a standard 2-square: $\{(\xi,\eta)| -1 \le \xi,\eta \le -1\}$.

Efficient quadrature coefficients are readily available in the literature so that any desired accuracy can readily be obtained.

From equation (35) we can write:

$$\begin{aligned} I &= \int_0^1 \int_0^1 f(x(\xi,\eta), y(\xi,\eta))(\frac{1-\xi}{8})\,\mathrm{d}\xi\,\mathrm{d}\eta, \\ I &= \sum_{i=1}^{n} \sum_{i=1}^{n} (\frac{1-\xi_i}{8}) w_i w_j f(x(\xi,\eta), y(\xi,\eta)), \end{aligned} \tag{36}$$

where ξ_i, η_i are Gaussian points in the ξ, η directions, respectively, and w_i and w_j are the corresponding weights.

We can rewrite equation (36) as:

$$I = \sum_{k=1}^{N=n\times n} c_k f(x_k, y_k), \tag{37}$$

where c_k, x_k and y_k can be obtained from the relations:

$$\begin{aligned} & c_k = \frac{1-\xi_i}{8} w_i w_j, \quad x_k = \frac{1+\xi_i}{2}, \quad y_k = \frac{(1-\xi_i)(1+\eta_i)}{4}, \\ & k,i,j = 1,2,3,\ldots,n. \end{aligned} \tag{38}$$

Here are the coefficients for $n = 2$:

x_i	y_i	c_i
0.211324865	0.166666667	0.197168783
0.211324865	0.622008467	0.197168783
0.788675134	0.044658198	0.052831216
0.788675134	0.166666667	0.052831216

0.8 Solver

I've implemented a sparse direct solver that uses Choletsky decomposition for symmetric systems.

Gibbs algorithm for finding a pseudo-peripheral node:

- (step1)(initialization):select any node r
- (step2)(build a structure of levels):build a structure of levels with root in r: L(r)={L[0](r),L[1](r),...,L[l(r)](r)}, where l(r) – excentricity of node r.
- (step3)(pulling down the last level): select from L[l(r)] a node x with the minimal degree
- (step4)(building a structure of levels):build a structure of levels with root in x: L(x)={L[0](x),L[1](x),...,L[l(x)](x)}
  if l(x)¿l(r), then put r=x and move to step 2.
- (step5) x is pseudo-peripheral

The structure of levels is built like this:
L[0](x)=x,L[1](x)=Adj(x),L[i](x)=Adj(L[i-1](x)-L[i-2](x)),i=2..l(x)

Reverse Cuthill-McKee algorithm for finding a symmetrical reordering of rows and columns, that decreases the profile of the system.

- (step1):Mark the starting node r and x[0]=r
- (step2)(main cycle):for i=0..N-1 find all unnumbered neighbors of x[i] and enumerate them in order of increasing of their degrees
- (step3)(reverse reordering):y[i]=x[N-i-1], i=0..N-1

If the graph is not linked, we use this algorithm on each linked part.

0.9 Sample problem

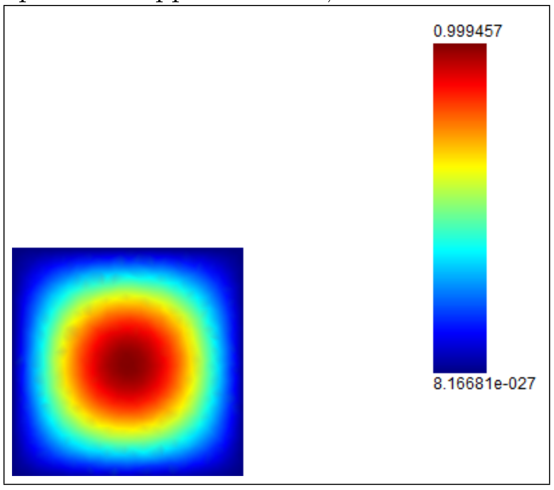
Let's see how the program works for this simple problem

$$
\begin{aligned}
\Delta u &= 2\pi^2 \sin(\pi x)\sin(\pi y) \text{ in } \Omega = [0,1]\times[0,1] \\
u &= 0 \text{ on } \partial\Omega
\end{aligned}
\tag{39}
$$

with known solution

$$
u = \sin(\pi x)\sin(\pi y) \tag{40}
$$

For quadratic approximation, the result looks like



Visually it looks correct, but to make sure, let's evaluate the error:

$$
e = ||u - u_h||_2 = \int_0^1 \int_0^1 (u - u_h)^2 \, \mathrm{d}x \, \mathrm{d}y \tag{41}
$$

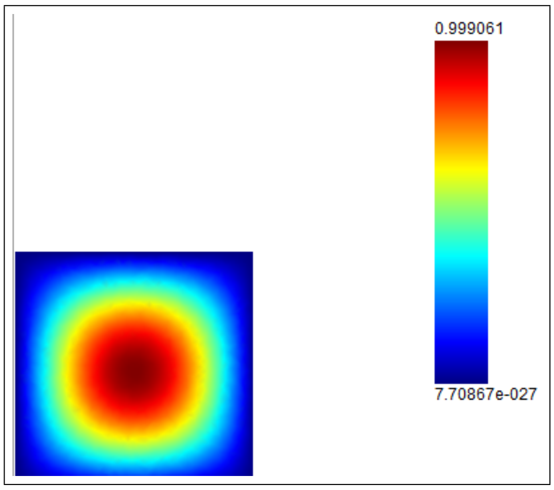
	$h = \frac{1}{10}$	$h = \frac{1}{20}$	rate m
linear	0.00414891	0.00104353	1.9913
quadratic	0.000101269	$1.19623{\cdot}10^{-5}$	3.0816

	$h = \frac{1}{20}$	$h = \frac{1}{40}$	rate my
linear	0.00104353	0.000261274	1.9978
quadratic	$1.19623{\cdot}10^{-5}$	$1.4637{\cdot}10^{-6}$	3.0310

Now let's change the heat conduction coefficient to this:

$$k = \begin{bmatrix} \frac{(x-0.5)^2}{\sqrt{(x-0.5)^2+(y-0.5)^2}} & \frac{(x-0.5)(y-0.5)}{\sqrt{(x-0.5)^2+(y-0.5)^2}} \\ \frac{(x-0.5)(y-0.5)}{\sqrt{(x-0.5)^2+(y-0.5)^2}} & \frac{(y-0.5)^2}{\sqrt{(x-0.5)^2+(y-0.5)^2}} \end{bmatrix} \tag{42}$$

We can use MATLAB to quickly compute what the corresponding right hand side should look like. The solution looks like this:



0.10 Time discretization

There is little difficulty in extending the finite element idealization to situations that are time dependant. By putting

$$u_h = \sum N_i a_i = \mathbf{Na}$$
$$N = N(x,y,z) \quad a = a(t) \tag{43}$$

for each element, then we get the following *matrix differential equation*:

$$\mathbf{C\dot{a}} + \mathbf{Ka} + \mathbf{f} = \mathbf{0} \tag{44}$$

$$\mathbf{a}(0) = \mathbf{a}_0 \tag{45}$$

in which all the matrices are assembled from element submatrices in the standard manner with **C** being the mass matrix and **K** being the stiffness matrix. To solve the problem we use the SS11 algorithm:

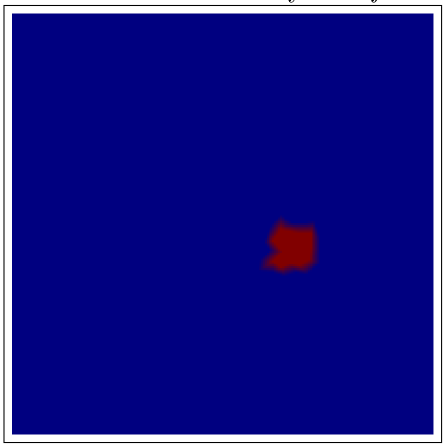
$$\mathbf{a_{n+1}} = \mathbf{a_n} + \mathbf{\Delta t}\alpha \tag{46}$$

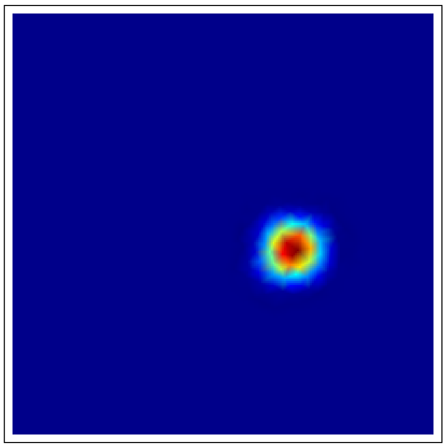
$$\alpha = -(\mathbf{C} + \theta\delta t\mathbf{K})^{-1}(f + \mathbf{Ka_n}) \tag{47}$$

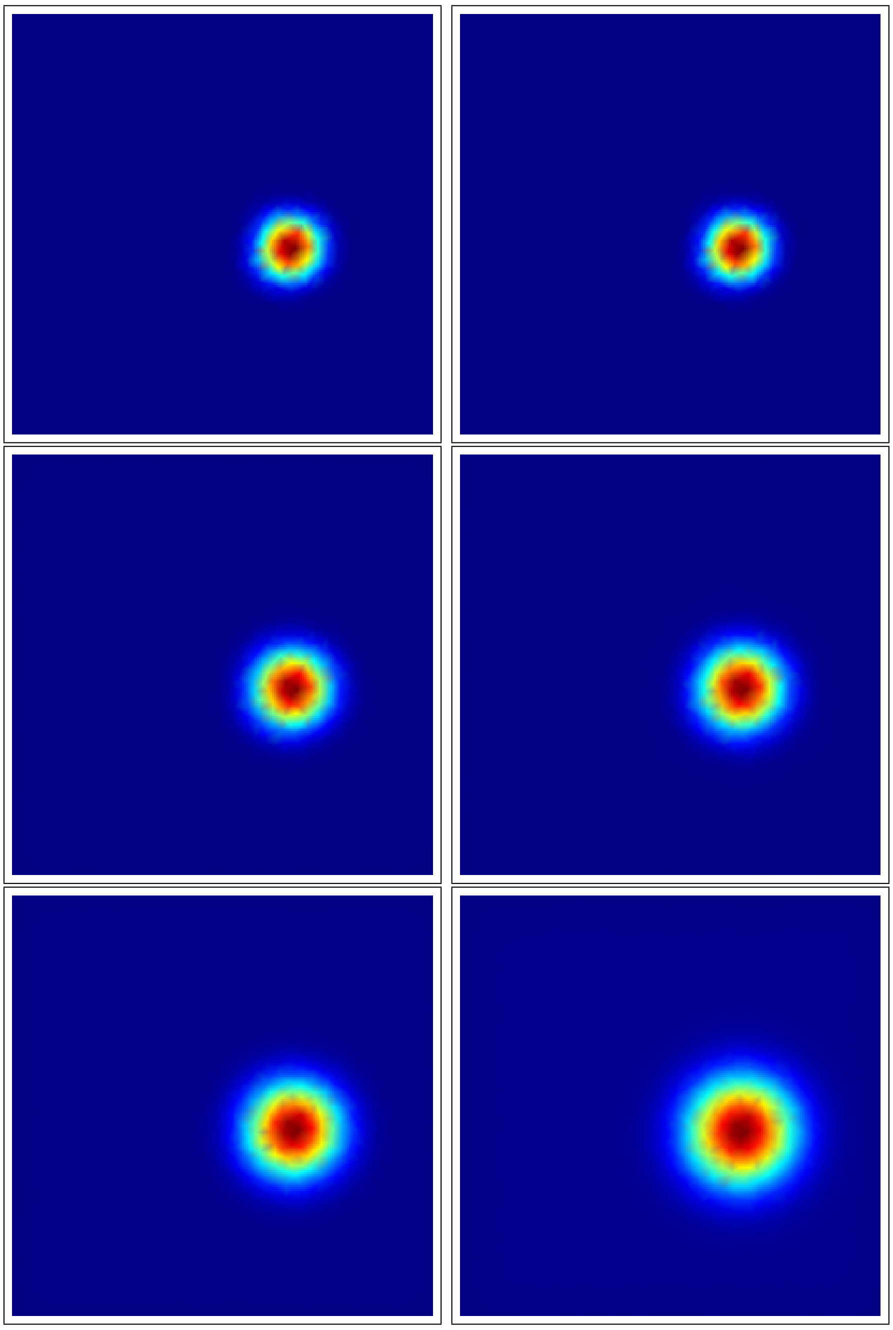
θ is chosen to be 0.5, which corresponds to Crank-Nicholson scheme.

0.10.1 Time dependant sample problem

This is result for k identity and f one.







```
#pragma once
#include "afxcmn.h"
#include "afxwin.h"
#include "TriangleDoc.h"

// CAddBoundaryView form view
            /*
             * This view lets you add and remove boundary
                conditions.
             * Use Ctrl+B to show/hide it.
             * Using the view is pretty straightforward.
             * After the boundary condition has been added,
                 it can be applied to any segment.
             */
class CAddBoundaryView : public CFormView
{
protected:
    CAddBoundaryView(); // create from serialization only
    DECLARE_DYNCREATE(CAddBoundaryView)
// Attributes
public:
    inline CTriangleDoc* GetDocument(){return (CTriangleDoc
        *)m_pDocument;};
// Operations

// Overrides
public:
    virtual void OnInitialUpdate();
    virtual void OnUpdate(CView* /*pSender*/, LPARAM /*
        lHint*/, CObject* /*pHint*/);
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    //
        DDX/DDV support
// Implementation
public:
    virtual ~CAddBoundaryView();
    enum { IDD = IDD_ADDBOUNDARYVIEW };
#ifdef _DEBUG
    virtual void AssertValid() const;
#ifndef _WIN32_WCE
    virtual void Dump(CDumpContext& dc) const;
#endif
#endif
protected:
    //...false before first InitialUpdate
    bool created;
    //...controls
    CComboBox m_comboType;
    CEdit m_editT0;
    CEdit m_editq0;
    CEdit m_edith;
    CEdit m_editT_inf;
    CButton m_btnAdd;
    CButton m_btnRemove;
    CButton m_btnApply;
    CListCtrl m_listBoundaries;
    int m_listBoundariesSel;     //number of item selected
       in list
    bool m_bEditing;             //if true, when edit text
       is changed, apply button is enabled
                                 //it's set to true when an
                                    item in list is selected

// Generated message map functions
protected:
    afx_msg void OnBnClickedbadd();
    afx_msg void OnBnClickedbremove();
    afx_msg void OnBnClickedbapply();
    afx_msg void OnCbnSelchangecombotype();
    afx_msg void OnLvnItemchangedlistboundaries(NMHDR *
       pNMHDR, LRESULT *pResult);
    afx_msg void OnNMKillfocuslistboundaries(NMHDR *pNMHDR,
        LRESULT *pResult);
    afx_msg void OnEnChangeEdits(UINT id);
    DECLARE_MESSAGE_MAP()
};
```

```
// AddBoundaryView.cpp : implementation file
//

#include "stdafx.h"
#include "MFCtriangle.h"
#include "AddBoundaryView.h"


// CAddBoundaryView

IMPLEMENT_DYNCREATE(CAddBoundaryView, CFormView)

CAddBoundaryView::CAddBoundaryView()
: CFormView(CAddBoundaryView::IDD),created(false),
    m_bEditing(false)
{

}

CAddBoundaryView::~CAddBoundaryView()
{
}

void CAddBoundaryView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_listBoundaries, m_listBoundaries);
    DDX_Control(pDX, IDC_comboType, m_comboType);
    DDX_Control(pDX, IDC_edit_T0, m_editT0);
    DDX_Control(pDX, IDC_edit_q0, m_editq0);
    DDX_Control(pDX, IDC_edit_h, m_edith);
    DDX_Control(pDX, IDC_edit_T_inf, m_editT_inf);
    DDX_Control(pDX, IDC_bApply, m_btnApply);
    DDX_Control(pDX, IDC_bRemove, m_btnRemove);
    DDX_Control(pDX, IDC_bAdd, m_btnAdd);
}

BEGIN_MESSAGE_MAP(CAddBoundaryView, CFormView)
    ON_BN_CLICKED(IDC_bAdd, &CAddBoundaryView::
        OnBnClickedbadd)
    ON_BN_CLICKED(IDC_bRemove, &CAddBoundaryView::
        OnBnClickedbremove)
    ON_CBN_SELCHANGE(IDC_comboType, &CAddBoundaryView::
        OnCbnSelchangecombotype)
    ON_CONTROL_RANGE(EN_CHANGE, IDC_edit_T0, IDC_edit_T_inf,&
        CAddBoundaryView::OnEnChangeEdits)
    ON_NOTIFY(LVN_ITEMCHANGED, IDC_listBoundaries, &
        CAddBoundaryView::OnLvnItemchangedlistboundaries)
    ON_NOTIFY(NM_KILLFOCUS, IDC_listBoundaries, &
        CAddBoundaryView::OnNMKillfocuslistboundaries)
    ON_BN_CLICKED(IDC_bApply, &CAddBoundaryView::
        OnBnClickedbapply)
END_MESSAGE_MAP()


// CAddBoundaryView diagnostics

#ifdef _DEBUG
void CAddBoundaryView::AssertValid() const
{
    CFormView::AssertValid();
}

#ifndef _WIN32_WCE
void CAddBoundaryView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}
#endif
#endif //_DEBUG


// CAddBoundaryView message handlers

void CAddBoundaryView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();
    m_editT0.EnableWindow(FALSE);
    m_editq0.EnableWindow(FALSE);
    m_edith.EnableWindow(FALSE);
    m_editT_inf.EnableWindow(FALSE);

    if(created==false)
    {
        created=true;
        m_comboType.AddString(_T("flux"));
        m_comboType.AddString(_T("temperature"));
        m_listBoundaries.ModifyStyle (LVS_TYPEMASK,
            LVS_REPORT);
        m_listBoundaries.InsertColumn(0,_T("T_inf"),
            LVCFMT_LEFT,50);
        m_listBoundaries.InsertColumn(0,_T("h"),LVCFMT_LEFT
            ,50);
        m_listBoundaries.InsertColumn(0,_T("q0"),
            LVCFMT_LEFT,50);
        m_listBoundaries.InsertColumn(0,_T("T0"),
            LVCFMT_LEFT,50);
        m_listBoundaries.InsertColumn(0,_T("N"),LVCFMT_LEFT
            ,30);
    }
    OnUpdate(0,0,0);
}
#define CEDITTOfloat(edit,value)    CEditTofloat(edit,value
    ,L#value)

BOOL CEditTofloat(CEdit& edit,float& value,wchar_t*
    parameter)
{
    wchar_t text[100];
    wchar_t dummy=0;wchar_t* end=&dummy;
    edit.GetWindowText(text,100);
    value=wcstod(text,&end);
    if(value==0)
    {
        //if(errno==EINVAL) this doesn't work for some
            reason
        if(end==text)
        {
            CString message;message.Format(_T("Parameter␣
                doesn't␣translate\nto␣a␣floating␣point␣value
                ␣%s!"),parameter);
            AfxMessageBox(message);
            edit.SetFocus();
            return TRUE;
        }
    }
    return FALSE;
}

void CAddBoundaryView::OnBnClickedbadd()
{
    CTriangleDoc* pDoc=GetDocument();
    switch(m_comboType.GetCurSel())
    {
    case 0://flux
        {
            float q0;
            if(CEDITTOfloat(m_editq0,q0))
                return;
            float h;
            if(CEDITTOfloat(m_edith,h))
                return;
            float T_inf;
            if(CEDITTOfloat(m_editT_inf,T_inf))
                return;
            pDoc->m_bndVector.push_back(new
                HeatFluxBoundary(q0,h,T_inf));
            OnUpdate(0,0,0);
        }break;
    case 1://temperature
        {
            float T0;
            if(CEDITTOfloat(m_editT0,T0))
                return;
            pDoc->m_bndVector.push_back(new
                TemperatureBoundary(T0));
            OnUpdate(0,0,0);
        }break;
    default://none
        {
            AfxMessageBox(_T("Select the boundary type 
                first!"));
            m_comboType.SetFocus();
        }
    }
    m_bEditing=false;
}

void CAddBoundaryView::OnBnClickedbremove()
{
    CTriangleDoc* pDoc=GetDocument();
    vector<gBoundary*>::iterator i=pDoc->m_bndVector.begin
        ();
    while(m_listBoundariesSel--)
        i++;
    delete (*i);
    pDoc->m_bndVector.erase(i);
    m_btnRemove.EnableWindow(FALSE);
    OnUpdate(0,0,0);
}

void CAddBoundaryView::OnCbnSelchangecombotype()
{
    CString s;
    m_comboType.GetLBText(m_comboType.GetCurSel(),s);
    if(s==_T("flux"))
    {
        m_editT0.EnableWindow(FALSE);
        m_editq0.EnableWindow(TRUE);
        m_edith.EnableWindow(TRUE);
        m_editT_inf.EnableWindow(TRUE);

        //m_editq0.SetFocus();
    }
    else
        if(s==_T("temperature"))
        {
            m_editT0.EnableWindow(TRUE);
            m_editq0.EnableWindow(FALSE);
            m_edith.EnableWindow(FALSE);
            m_editT_inf.EnableWindow(FALSE);

            //m_editT0.SetFocus();
        }
        else
            ;
}

void CAddBoundaryView::OnUpdate(CView* /*pSender*/, LPARAM
    /*lHint*/, CObject* /*pHint*/)
{
    CTriangleDoc* pDoc=GetDocument();
    m_listBoundaries.DeleteAllItems();

    CString string;
    vector<gBoundary*>&a=pDoc->m_bndVector;
    int n=a.size();
    for(int i=0;i<n;i++)
    {
        string.Format(_T("%d"),i);
        m_listBoundaries.InsertItem(i,string);

        m_listBoundaries.SetItemText(i,1,a[i]->Info(0));
        m_listBoundaries.SetItemText(i,2,a[i]->Info(1));
        m_listBoundaries.SetItemText(i,3,a[i]->Info(2));
        m_listBoundaries.SetItemText(i,4,a[i]->Info(3));
    }
}

void CAddBoundaryView::OnLvnItemchangedlistboundaries(NMHDR
     *pNMHDR, LRESULT *pResult)
{
    LPNMLISTVIEW pNMLV = reinterpret_cast<LPNMLISTVIEW>(
        pNMHDR);
    if((pNMLV->uChanged==LVIF_STATE)&&(pNMLV->uNewState))
    {
        CTriangleDoc* pDoc=GetDocument();
        gBoundary* a=pDoc->m_bndVector[m_listBoundariesSel=
            pNMLV->iItem];
        m_editT0.SetWindowText(a->Info(0));
        m_editq0.SetWindowText(a->Info(1));
        m_edith.SetWindowText(a->Info(2));
        m_editT_inf.SetWindowText(a->Info(3));
        if(a->Info(0)==_T("-"))
            m_comboType.SetCurSel(0);
        else
            m_comboType.SetCurSel(1);
        OnCbnSelchangecombotype();
        if(m_listBoundariesSel>1)//can't remove two basic
            boundaries: zero temperature and insulation
            m_btnRemove.EnableWindow(TRUE);
        else
            m_btnRemove.EnableWindow(FALSE);
        m_btnApply.EnableWindow(FALSE);//the parameters are
             unchanged here, nothing to apply
        m_bEditing=true;
    }
    *pResult = 0;
}

void CAddBoundaryView::OnNMKillfocuslistboundaries(NMHDR *
   pNMHDR, LRESULT *pResult)
{
    if(GetFocus()!=&m_btnRemove)
        m_btnRemove.EnableWindow(FALSE);
    *pResult = 0;
}

void CAddBoundaryView::OnEnChangeEdits(UINT id)
{
    if(m_bEditing)
        m_btnApply.EnableWindow(TRUE);
}

void CAddBoundaryView::OnBnClickedbapply()
{
    CTriangleDoc* pDoc=GetDocument();
    vector<gBoundary*>& a=pDoc->m_bndVector;
    vector<gBoundary*>::iterator i=a.begin()+
        m_listBoundariesSel;
    gBoundary* newItem;
    switch(m_comboType.GetCurSel())
    {
    case 0://flux
        {
            float q0;
            if(CEDITTOfloat(m_editq0,q0))
                return;
            float h;
            if(CEDITTOfloat(m_edith,h))
                return;
            float T_inf;
            if(CEDITTOfloat(m_editT_inf,T_inf))
                return;
            newItem=new HeatFluxBoundary(q0,h,T_inf);
        }break;
    case 1://temperature
        {
            float T0;
            if(CEDITTOfloat(m_editT0,T0))
                return;
            newItem=new TemperatureBoundary(T0);
        }break;
    default://none
        _ASSERTE(1);
    }
    i=++a.insert(i,newItem);
    delete (*i);
    a.erase(i);
    m_btnApply.EnableWindow(FALSE);
    OnUpdate(0,0,0);
}
```

```
#pragma once
#include "afxwin.h"


// CEllipseDialog dialog

class CEllipseDialog : public CDialog
{
    DECLARE_DYNAMIC(CEllipseDialog)

public:
    CEllipseDialog(CWnd* pParent = NULL);   // standard
       constructor
    virtual ~CEllipseDialog();

// Dialog Data
    enum { IDD = IDD_ELLIPSEDIALOG };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);     //
       DDX/DDV support

    DECLARE_MESSAGE_MAP()
public:
    double Left;
    double Top;
    double Right;
    double Bottom;
};
```

```
// EllipseDialog.cpp : implementation file
//

#include "stdafx.h"
#include "MFCtriangle.h"
#include "EllipseDialog.h"


// CEllipseDialog dialog

IMPLEMENT_DYNAMIC(CEllipseDialog, CDialog)

CEllipseDialog::CEllipseDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CEllipseDialog::IDD, pParent)
	,Left(0),Top(0)

	, Right(0)
	, Bottom(0)
{
}

CEllipseDialog::~CEllipseDialog()
{
}

void CEllipseDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Text(pDX, IDC_EDITleft, Left);
	DDX_Text(pDX, IDC_EDITtop, Top);
	DDX_Text(pDX, IDC_EDITright, Right);
	DDX_Text(pDX, IDC_EDITbottom, Bottom);

}


BEGIN_MESSAGE_MAP(CEllipseDialog, CDialog)
END_MESSAGE_MAP()


// CEllipseDialog message handlers
```

```
#pragma once
//this is an edit class that accepts only floating point
   number input
//and sends WM_ENTEREDIT message to it's parent when Enter
   key is pressed
#define WM_ENTEREDIT            WM_USER+0x100

class CEnterEdit:public CEdit
{
public:
    CEnterEdit():dot(false){}
protected:
    bool dot;
    afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT
        nFlags);
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT
        nFlags);
    afx_msg void OnEnChange();
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    DECLARE_MESSAGE_MAP()
};
```

```
#include "stdafx.h"
#include "EnterEdit.h"
BEGIN_MESSAGE_MAP(CEnterEdit, CEdit)
     ON_WM_CHAR()
     ON_WM_KEYDOWN()
     ON_WM_SETFOCUS()
     ON_CONTROL_REFLECT(EN_CHANGE, &CEnterEdit::OnEnChange)
END_MESSAGE_MAP()

void CEnterEdit::OnChar(UINT nChar, UINT nRepCnt, UINT
    nFlags)
{
     switch(nChar)
     {
     case VK_TAB:
          {
               CEdit* next=(CEdit*)GetWindowLong(m_hWnd,
                   GWL_USERDATA);
               if(next)
                    next->SetFocus();
               return;
          }break;
     case _T('-'):
          {
               CString string;GetWindowText(string);
               if(!string.IsEmpty())
                    return;
          }break;
     case _T('.'):
          if(dot)
               return;
          dot=true;
          break;
     case VK_BACK:
          {
          }break;
     default:
          if ((nChar<_T('0'))||(nChar>_T('9')))
               return;
     }
     CEdit::OnChar(nChar, nRepCnt, nFlags);
}

void CEnterEdit::OnKeyDown(UINT nChar, UINT nRepCnt, UINT
    nFlags)
{
     if(nChar==13)
     {
          CString string;GetWindowText(string);
```

```
        if(string.IsEmpty())
            return;
        GetParent()->SendMessage(WM_ENTEREDIT);
        return;
    }
    CEdit::OnKeyDown(nChar, nRepCnt, nFlags);
}

void CEnterEdit::OnSetFocus(CWnd* pOldWnd)
{
    CEdit::OnSetFocus(pOldWnd);
    //select all
    SetSel(0,-1,TRUE);
}
void CEnterEdit::OnEnChange()
{
    CString string;GetWindowText(string);
    if(string.Find(_T("."))>=0)
        dot=true;
    else
        dot=false;
}
```

```
// MainFrm.h : interface of the CMainFrame class
//
#include "TriangleDoc.h"
#include "TriangleView.h"
#include "EnterEdit.h"
#pragma once
/////////////////////////////
#define WM_UPDATEPOINTTRACKER    WM_USER+0x101
//this is a class that correlates two edit controls
//to point with two floating point coordinates
class CPointTracker : public CWnd
{
protected:
    UINT nID;
    CEnterEdit m_editX;
    CEnterEdit m_editY;
public:
    CPointTracker(){};
    virtual ~CPointTracker(){};
    BOOL Create(DWORD dwStyle,const RECT& rect,CWnd*
        pParentWnd,UINT nid);
    void SetPoint(gPOINT point);
protected:
    afx_msg LRESULT OnEnterEdit(WPARAM wParam,LPARAM lParam
        );
    DECLARE_MESSAGE_MAP()
};
/////////////////////////////
class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
// Operations
public:
    int GetPolySides()const{return m_wndPolySpin.GetPos();}
// Overrides
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs,
        CCreateContext* pContext);
// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
```

```
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    CToolBar m_wndToolBar;
        CSpinButtonCtrl m_wndPolySpin;
        CEdit m_wndPolyEdit;

    CStatusBar m_wndStatusBar;
        CPointTracker m_wndPointTracker;
public:
    CSplitterWnd m_wndSplitter;
    CSplitterWnd m_wndSplitterInfo;
// Generated message map functions
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg LRESULT OnUpdatePointTracker(WPARAM wParam,
       LPARAM lParam);
    DECLARE_MESSAGE_MAP()
    bool boundarysettings;
    bool indexsettings;
public:
    afx_msg void OnOptionsBoundarysettings();
    afx_msg void OnUpdateOptionsBoundarysettings(CCmdUI *
       pCmdUI);
public:
    afx_msg void OnOptionsIndexes();
public:
    afx_msg void OnUpdateOptionsIndexes(CCmdUI *pCmdUI);
public:
    afx_msg void OnFileTrianglulate();
public:
    afx_msg void OnFileSolve();
public:
    afx_msg void OnOptionsPreferences();
public:
    afx_msg void OnOptionsSubdomainsettings();
public:
    afx_msg void OnOptionsFocus();
public:
    afx_msg void OnFileSolvetimedependant();
public:
    afx_msg void OnFileExportastext();
    afx_msg void OnApproximationType(UINT);
    afx_msg void OnUpdateApproximationType(CCmdUI* pCmdUI);
    afx_msg void OnQuadratureType(UINT);
    afx_msg void OnUpdateQuadratureType(CCmdUI* pCmdUI);
    afx_msg void OnInsertElliplse();
```

```
};
```

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "MFCtriangle.h"
//#include "TrianglePointsView.h"
//#include "SegmentsView.h"
//#include "TriIndexView.h"
#include "PrefDialog.h"
#include "AddBoundaryView.h"
#include "SubdomainDialog.h"
#include "MainFrm.h"
#include "SolveDialog.h"
#include "EllipseDialog.h"
#include <fstream>

using namespace std;

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	ON_WM_CREATE()
	ON_MESSAGE(WM_UPDATEPOINTTRACKER, OnUpdatePointTracker)
	ON_WM_CTLCOLOR()
	ON_COMMAND(ID_OPTIONS_BOUNDARYSETTINGS, &CMainFrame::
		OnOptionsBoundarysettings)
	ON_UPDATE_COMMAND_UI(ID_OPTIONS_BOUNDARYSETTINGS, &
		CMainFrame::OnUpdateOptionsBoundarysettings)
	ON_COMMAND(ID_OPTIONS_INDEXES, &CMainFrame::
		OnOptionsIndexes)
	ON_UPDATE_COMMAND_UI(ID_OPTIONS_INDEXES, &CMainFrame::
		OnUpdateOptionsIndexes)
	ON_COMMAND(ID_FILE_TRIANGLULATE, &CMainFrame::
		OnFileTrianglulate)
	ON_COMMAND(ID_FILE_SOLVE, &CMainFrame::OnFileSolve)
	ON_COMMAND(ID_OPTIONS_PREFERENCES, &CMainFrame::
		OnOptionsPreferences)
	ON_COMMAND(ID_OPTIONS_SUBDOMAINSETTINGS, &CMainFrame::
		OnOptionsSubdomainsettings)
	ON_COMMAND(ID_OPTIONS_FOCUS, &CMainFrame::
		OnOptionsFocus)
	ON_COMMAND(ID_FILE_SOLVETIMEDEPENDANT, &CMainFrame::
```

```
            OnFileSolvetimedependant)
        ON_COMMAND(ID_FILE_EXPORTASTEXT, &CMainFrame::
            OnFileExportastext)
        ON_COMMAND_RANGE(ID_APPROXIMATION_LINEAR,
            ID_APPROXIMATION_ISOPARAMETRIC, CMainFrame::
            OnApproximationType)
        ON_COMMAND_RANGE(ID_QUADRATURE_GAUSS2,
            ID_QUADRATURE_GAUSS3, CMainFrame::OnQuadratureType)
        ON_UPDATE_COMMAND_UI_RANGE(ID_APPROXIMATION_LINEAR,
            ID_APPROXIMATION_ISOPARAMETRIC, CMainFrame::
            OnUpdateApproximationType)
        ON_UPDATE_COMMAND_UI_RANGE(ID_QUADRATURE_GAUSS2,
            ID_QUADRATURE_GAUSS3, CMainFrame::
            OnUpdateQuadratureType)
        ON_COMMAND(ID_INSERT_ELLIPLSE, &CMainFrame::
            OnInsertElliplse)
END_MESSAGE_MAP()


// CMainFrame construction/destruction

CMainFrame::CMainFrame():boundarysettings(false),
    indexsettings(false)
{
}

CMainFrame::~CMainFrame()
{
}


BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    return TRUE;
}


// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG


// CMainFrame message handlers
/////////////////////////////
//this function loads bitmaps from files into the imagelist
// n is number of bitmaps
// pszFileNames is the array of file paths
void LoadToolBarImages(int n,TCHAR** pszFileNames,
    CImageList* pList)
{
    for(int i=0;i<n;i++)
    {
        HBITMAP hBitmap=(HBITMAP)::LoadImage(0,pszFileNames
            [i],IMAGE_BITMAP,16,15,LR_LOADFROMFILE);
        CBitmap bitmap;
        bitmap.Attach(hBitmap);
        pList->Add(&bitmap,RGB(0,0,0));
        bitmap.Detach();
    }
}
/////////////////////////////
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    m_wndToolBar.CreateEx(this,TBSTYLE_FLAT, WS_CHILD |
        WS_VISIBLE | CBRS_TOP
         | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
            CBRS_SIZE_DYNAMIC);//TBSTYLE_FLAT|
            TBSTYLE_TOOLTIPS|WS_CLIPCHILDREN,CBRS_GRIPPER);
    m_wndToolBar.SetWindowText(_T("Standard"));
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);
    //FloatControlBar(&m_wndToolBar,CPoint(50,50));

    CImageList iList;iList.Create(16,15,ILC_COLOR32,0,10);
        //contains 32-bit bitmaps
    static TCHAR* Images[6]={_T("images\\New.bmp"),_T("
        images\\Open.bmp"),_T("images\\Save.bmp"),_T("images
        \\Poly.bmp"),_T("images\\Triangle.bmp"),_T("images\\
        Solve.bmp")};
    LoadToolBarImages(6,Images,&iList);
    m_wndToolBar.GetToolBarCtrl().SetImageList(&iList);
    iList.Detach();
    iList.Create(16,15,ILC_COLOR32,0,10);
    static TCHAR* HotImages[6]={_T("images\\hot\\New.bmp"),
        _T("images\\hot\\Open.bmp"),_T("images\\hot\\Save.
        bmp"),_T("images\\hot\\Poly.bmp"),_T("images\\hot\\
        Triangle.bmp"),_T("images\\hot\\Solve.bmp")};
    LoadToolBarImages(6,HotImages,&iList);
    m_wndToolBar.GetToolBarCtrl().SetHotImageList(&iList);
    iList.Detach();

    UINT nButtonIDs[]={ID_FILE_NEW,ID_FILE_OPEN,
        ID_FILE_SAVE,ID_SEPARATOR,ID_INSERT_POLYGON,
        ID_SEPARATOR,ID_SEPARATOR,ID_FILE_TRIANGLULATE,
        ID_FILE_SOLVE};
    m_wndToolBar.SetButtons(nButtonIDs,9);

    m_wndToolBar.SetButtonStyle(4,TBBS_CHECKBOX);//poly
    m_wndToolBar.SetButtonStyle(7,TBBS_CHECKBOX);//
        triangulate

    //to insert updown and edit controls add an
        ID_SEPARATOR id,then resize the item and create
        controls as children of toolbar
    //and place them in the separator's new position
    m_wndToolBar.SetButtonInfo(5,IDC_UPDOWNWITHEDIT,
        TBBS_SEPARATOR,70);
    CRect rect;m_wndToolBar.GetItemRect(5,&rect);
    rect.bottom=rect.top+20;
    rect.left+=23;
    m_wndPolyEdit.Create(WS_CHILD|WS_VISIBLE,rect,&
        m_wndToolBar,IDC_POLYEDIT);
    CRect srect=rect;
    srect.right=srect.left-1;
    srect.left-=20;
    m_wndPolySpin.Create(WS_CHILD|WS_VISIBLE|UDS_AUTOBUDDY|
        UDS_SETBUDDYINT,srect,&m_wndToolBar,IDC_POLYSPIN);
    m_wndPolySpin.SetBuddy(&m_wndPolyEdit);
    m_wndPolyEdit.MoveWindow(&rect);
    m_wndPolySpin.SetRange(3,1000);
    m_wndPolySpin.SetPos(5);


    m_wndStatusBar.Create(this);
    m_wndPointTracker.Create(0,CRect(1,1,180,17),&
        m_wndStatusBar,0);
    UINT nIndicators[]={ID_SEPARATOR,ID_SEPARATOR};
    m_wndStatusBar.SetIndicators(nIndicators,2);
    m_wndStatusBar.SetPaneInfo(0,1,0,180);
    m_wndStatusBar.SetPaneInfo(1,2,0,180);
    m_wndStatusBar.SetPaneText(1,_T(""));
```

```
    return 0;
}
/////////////////////////////
afx_msg LRESULT CMainFrame::OnUpdatePointTracker( WPARAM
   wParam,LPARAM lParam )
{
    if(wParam==1)
    {
        double* v=(double*)lParam;
        CString s;s.Format(_T("%g"),*v);
        m_wndStatusBar.SetPaneText(1,s);
    }
    else
        m_wndPointTracker.SetPoint(*( (gPOINT*)lParam ));
    return 0;
}
//CPointTracker!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
BEGIN_MESSAGE_MAP(CPointTracker,CWnd)
    ON_MESSAGE(WM_ENTEREDIT, OnEnterEdit)
END_MESSAGE_MAP()

BOOL CPointTracker::Create( DWORD dwStyle,const RECT& rect,
   CWnd* pParentWnd,UINT nid )
{
    CString string=AfxRegisterWndClass(0,LoadCursor(
        AfxGetInstanceHandle(),IDC_CROSS),(HBRUSH)
        GetStockObject(GRAY_BRUSH));

    nID=nid;
    CreateEx(0,string,0,WS_VISIBLE|WS_CHILD|dwStyle,rect,
        pParentWnd,nID);
    m_editX.Create(WS_VISIBLE|WS_CHILD|ES_AUTOHSCROLL|
        ES_WANTRETURN|ES_MULTILINE,
         CRect(1,1,CRect(rect).Width()/2-1,rect.bottom),this
             ,1);
    m_editY.Create(WS_VISIBLE|WS_CHILD|ES_AUTOHSCROLL|
        ES_WANTRETURN|ES_MULTILINE,
         CRect(CRect(rect).Width()/2+1,1,CRect(rect).Width()
             ,rect.bottom),this,2);
    ::SetWindowLong(m_editX.m_hWnd,GWL_USERDATA,(LONG)&
        m_editY);//used for tabbing between edits
    ::SetWindowLong(m_editY.m_hWnd,GWL_USERDATA,(LONG)&
        m_editX);
    return 0;
}

afx_msg LRESULT CPointTracker::OnEnterEdit( WPARAM wParam,
   LPARAM lParam )
{
    CString string;
    m_editX.GetWindowText(string);
    double x=_tstof(string);
    m_editY.GetWindowText(string);
    double y=_tstof(string);

    CView* activeview=((CMainFrame*)AfxGetMainWnd())->
        GetActiveView();
    if(activeview->IsKindOf(RUNTIME_CLASS(CTriangleView)))
        ((CTriangleView*)(activeview))->UpdateCurrentVertex
            (gPOINT(x,y));
    else
        AfxMessageBox(_T("Select appropriate view first!"))
            ;
    return 0;
}

void CPointTracker::SetPoint( gPOINT point )
{
    CString string;
    string.Format(_T("%f"),point.x);
    m_editX.SetWindowText(string);
    string.Format(_T("%f"),point.y);
    m_editY.SetWindowText(string);
}

BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
    CCreateContext* pContext)
{
    CRect rect;GetClientRect(&rect);
    if(!m_wndSplitter.CreateStatic(this,1,2)||
        !m_wndSplitter.CreateView(0,0,RUNTIME_CLASS(
            CAddBoundaryView),CSize(/*280*/1,0),pContext)||
        !m_wndSplitter.CreateView(0,1,RUNTIME_CLASS(
            CTriangleView),CSize(rect.Width()/*-300*/,0),
            pContext))
        return FALSE;
    return TRUE;
}

void CMainFrame::OnOptionsBoundarysettings()
{
    CRect rect;GetClientRect(&rect);
    if(boundarysettings)
    {
        m_wndSplitter.SetColumnInfo(0,0,0);
        CView* wnd=(CView*)m_wndSplitter.GetPane(0,1);
        ASSERT(wnd->IsKindOf(RUNTIME_CLASS(CTriangleView)))
            ;
        wnd->SetFocus();
    }
    else
    {
        m_wndSplitter.SetColumnInfo(0,260,140);
        CView* wnd=(CView*)m_wndSplitter.GetPane(0,0);
        ASSERT(wnd->IsKindOf(RUNTIME_CLASS(CAddBoundaryView
            )));
        wnd->SetFocus();
    }
    m_wndSplitter.RecalcLayout();
    boundarysettings=!boundarysettings;
}

void CMainFrame::OnUpdateOptionsBoundarysettings(CCmdUI *
    pCmdUI)
{
    pCmdUI->SetCheck(boundarysettings);
}

void CMainFrame::OnOptionsIndexes()
{
    CRect rect;GetClientRect(&rect);
    if(indexsettings)
    {
        m_wndSplitter.SetColumnInfo(2,0,1000);
    }
    else
    {
        m_wndSplitter.SetColumnInfo(2,260,0);
        if(boundarysettings)
            m_wndSplitter.SetColumnInfo(1,rect.Width()
                -260*2,0);
        else
            m_wndSplitter.SetColumnInfo(1,rect.Width()
                -260,0);
    }
    m_wndSplitter.RecalcLayout();
    indexsettings=!indexsettings;
}

void CMainFrame::OnUpdateOptionsIndexes(CCmdUI *pCmdUI)
{
    pCmdUI->SetCheck(indexsettings);
}

void CMainFrame::OnFileTrianglulate()
{
    CWnd* view=m_wndSplitter.GetPane(0,1);
    _ASSERTE(view->IsKindOf(RUNTIME_CLASS(CTriangleView)));
    ((CTriangleView*)(view))->OnFileTrianglulate();
}

void CMainFrame::OnFileSolve()
{
    CWnd* view=m_wndSplitter.GetPane(0,1);
    _ASSERTE(view->IsKindOf(RUNTIME_CLASS(CTriangleView)));
    ((CTriangleView*)(view))->OnFileSolve();
}

void CMainFrame::OnOptionsPreferences()
{
    CWnd* view=m_wndSplitter.GetPane(0,1);
    _ASSERTE(view->IsKindOf(RUNTIME_CLASS(CTriangleView)));
    ((CTriangleView*)(view))->OnOptionsPreferences();
}

void CMainFrame::OnOptionsSubdomainsettings()
{
    CTriangleDoc* pDoc=(CTriangleDoc*)GetActiveDocument();
    CSubdomainDialog dlg;

    if(dlg.DoModal()==IDOK){}
}

void CMainFrame::OnOptionsFocus()
{
    CView* wnd=(CView*)m_wndSplitter.GetPane(0,1);
    ASSERT(wnd->IsKindOf(RUNTIME_CLASS(CTriangleView)));
    wnd->SetFocus();
}

void CMainFrame::OnFileSolvetimedependant()
{
    CSolveDialog dlg;
    if(dlg.DoModal()==IDOK)
    {}
}

void CMainFrame::OnFileExportastext()
{
    //CString fileName;
    //CFileDialog fileDialog(FALSE);
    //if(fileDialog.DoModal()==IDOK)
    //  fileName=fileDialog.GetPathName();
    //else
    //  return;
    //CTriangleDoc* pDoc=(CTriangleDoc*)GetActiveDocument()
    ;
//wchar_t filename[255];
//_tcscpy_s(filename,255,fileName);
//
//wofstream out(filename);
////save file name
//fileName=pDoc->GetTitle();
//_tcscpy_s(filename,255,fileName);
//out<<"//NAME="<<filename<<"//\n";
//list<gPOINT>* initial=&pDoc->m_ptList;;
///*if(pDoc->m_bTri)
//  initial=&pDoc->m_ptBkList;*/
////save point data
//out<<"//INITIAL MESH//\n";
//list<gPOINT>::iterator i=initial->begin(),end=initial
    ->end();
//if(i==end)
//  out<<"(empty)\n";
//out.setf(ios::scientific);
//out.precision(7);
//int j=0;
//for(;i!=end;++i)
//  out<<++j<<":\tx="<<(*i).x<<"\ty="<<(*i).y<<"\
    tborder="<<(*i).border()-10<<"\n";
//out<<"//TRIANGULATED MESH VERTICES//\n";

//vector<gPOINT>& tr=pDoc->m_ptVector;
//int n=tr.size();
//if(n==0)
//  out<<"(empty)\n";
//for(int i=0;i<n;++i)
//{
//  out<<i+1<<":\tx="<<tr[i].x;
//  out<<"\ty="<<tr[i].y;
//  if(tr[i].border())
//      out<<"\tborder="<<tr[i].border()-10;
//  else
//      out<<"\t(not on border)";
//  out<<"\n";
//}
//out<<"//TRIANGULATED MESH TRIANGLES//\n";
//list<CTriangle>& tris=pDoc->m_triList;
//if(pDoc->m_bTri)
//{
//  list<CTriangle>::iterator i=tris.begin(),end=tris.
    end();
//  int j=0;
//  for(;i!=end;++i)
//      out<<++j<<":\t"<<(*i).i1<<" "<<(*i).i2<<" "<<(*
        i).i3<<"\n";
    //}
    //else
    //  out<<"(empty)\n";
    //out<<"//TRIANGULATED MESH SEGMENTS//\n";
    //list<CSegment>& segs=pDoc->m_segList;
    //if(pDoc->m_bTri)
    //{
    //  list<CSegment>::iterator i=segs.begin(),end=segs.
        end();
    //  int j=0;
    //  for(;i!=end;++i)
    //      out<<++j<<":\t"<<(*i).x<<" "<<(*i).y<<"
        boundary="<<(*i).boundary-10<<"\n";
    //}
    //else
    //  out<<"(empty)\n";
    //n=pDoc->U.size();
    //if(n==0)
    //  return;
    //for(int i=0;i<n;++i)
    //{
    //  out<<"//TIME DEPENDENT SOLUTION: LAYER "<<i+1<<"//\
        n";
    //  int m=pDoc->m_ptVector.size();
    //  for(int j=0;j<m;++j)
    //      out<<j+1<<":\t"<<pDoc->U[i][j]<<"\n";
    //}
}

void CMainFrame::OnApproximationType(UINT nID)
{
    switch(nID)
    {
    case ID_APPROXIMATION_LINEAR:
        CTriangle::type=CTriangle::linear;break;
    case ID_APPROXIMATION_QUADRATIC:
        CTriangle::type=CTriangle::quadratic;break;
    case ID_APPROXIMATION_ISOPARAMETRIC:
        CTriangle::type=CTriangle::isoparametric;break;
    }
}

void CMainFrame::OnUpdateApproximationType(CCmdUI* pCmdUI)
{
    switch(pCmdUI->m_nID)
    {
    case ID_APPROXIMATION_LINEAR:
        pCmdUI->SetRadio(CTriangle::type==CTriangle::linear
            );break;
    case ID_APPROXIMATION_QUADRATIC:
        pCmdUI->SetRadio(CTriangle::type==CTriangle::
            quadratic);break;
    case ID_APPROXIMATION_ISOPARAMETRIC:
        pCmdUI->SetRadio(CTriangle::type==CTriangle::
            isoparametric);break;
    }
}

void CMainFrame::OnQuadratureType(UINT nID)
{
    switch(nID)
    {
    case ID_QUADRATURE_GAUSS2:
        CTriangle::qtype=CTriangle::gauss2;break;
    case ID_QUADRATURE_GAUSS3:
        CTriangle::qtype=CTriangle::gauss3;break;
    }
}

void CMainFrame::OnUpdateQuadratureType(CCmdUI* pCmdUI)
{
    switch(pCmdUI->m_nID)
    {
    case ID_QUADRATURE_GAUSS2:
        pCmdUI->SetRadio(CTriangle::qtype==CTriangle::
            gauss2);break;
    case ID_QUADRATURE_GAUSS3:
        pCmdUI->SetRadio(CTriangle::qtype==CTriangle::
            gauss3);break;
    }
}

void CMainFrame::OnInsertElliplse()
{
    /*task:
    create a spline bounded domain, based on outer normals
    create an ellipse: (x-x0)^2/a^2+(y-y0)^2/b^2=1
    create an ellipse based on general addition routine,
       that takes points and normals as input
    so this funciton basically generates the points and
       normals and calls the general one to create the
       domain
    */
    CTriangleDoc* pDoc=(CTriangleDoc*)GetActiveDocument();
    CEllipseDialog dlg;
    dlg.Left=pDoc->m_Left;
    dlg.Right=pDoc->m_Right;
    dlg.Top=pDoc->m_Top;
    dlg.Bottom=pDoc->m_Bottom;
    if(dlg.DoModal()==IDOK)
    {
        double a=abs(dlg.Right-dlg.Left)/2;
        double b=abs(dlg.Bottom-dlg.Top)/2;
        double x0=(dlg.Left+dlg.Right)/2;
        double y0=(dlg.Top+dlg.Bottom)/2;
        const int parts=7;
        double h=PI2/parts;
        for(double t=0;t<PI2;t+=h)
            pDoc->domain.points.push_back(gPOINT(x0+a*cos(t
                ),y0+b*sin(t)));
        pDoc->domain.currVertex=pDoc->domain.points.begin()
            ;
        for(double t=0;t<PI2;t+=h)
        {
            double n1x=b*cos(t),n1y=a*sin(t);
            double n2x=b*cos(t-h),n2y=a*sin(t-h);

            double x1=x0+a*cos(t),y1=y0+b*sin(t);
            double x2=x0+a*cos(t-h),y2=y0+b*sin(t-h);

            double x3=(x1*n1x*n2y-x2*n2x*n1y+n1y*n2y*(y1-y2
                ))/(n1x*n2y-n2x*n1y);
            double y3=(y1*n1y*n2x-y2*n2y*n1x+n1x*n2x*(x1-x2
                ))/(n1y*n2x-n2y*n1x);

            pDoc->domain.SetMidpoint(gPOINT(x3,y3));
            pDoc->domain.NextVertex();
        }
        pDoc->UpdateAllViews(0);
    }
}
```

```
// MFCtriangle.h : main header file for the MFCtriangle
    application
//
#pragma once

#ifndef __AFXWIN_H__
    #error "include␣'stdafx.h'␣before␣including␣this␣file␣
        for␣PCH"
#endif

#include "resource.h"        // main symbols


// CTriangleApp:
// See MFCtriangle.cpp for the implementation of this class
//

class CTriangleApp : public CWinApp
{
public:
    CTriangleApp();


// Overrides
public:
    virtual BOOL InitInstance();

// Implementation
    afx_msg void OnAppAbout();
    DECLARE_MESSAGE_MAP()
};

extern CTriangleApp theApp;
```

```
// MFCtriangle.cpp : Defines the class behaviors for the
   application.
//

#include "stdafx.h"
#include "MFCtriangle.h"
#include "MainFrm.h"

#include "TriangleDoc.h"
#include "TriangleView.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CTriangleApp

BEGIN_MESSAGE_MAP(CTriangleApp, CWinApp)
    ON_COMMAND(ID_APP_ABOUT, &CTriangleApp::OnAppAbout)
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, &CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, &CWinApp::OnFileOpen)
END_MESSAGE_MAP()


// CTriangleApp construction

CTriangleApp::CTriangleApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}


// The one and only CTriangleApp object

CTriangleApp theApp;


// CTriangleApp initialization

BOOL CTriangleApp::InitInstance()
{
    // InitCommonControlsEx() is required on Windows XP if
        an application
    // manifest specifies use of ComCtl32.dll version 6 or
        later to enable
```

```
// visual styles.  Otherwise, any window creation will
    fail.
INITCOMMONCONTROLSEX InitCtrls;
InitCtrls.dwSize = sizeof(InitCtrls);
// Set this to include all the common control classes
    you want to use
// in your application.
InitCtrls.dwICC = ICC_WIN95_CLASSES;
InitCommonControlsEx(&InitCtrls);

CWinApp::InitInstance();

// Standard initialization
// If you are not using these features and wish to
    reduce the size
// of your final executable, you should remove from the
     following
// the specific initialization routines you do not need
// Change the registry key under which our settings are
     stored
// TODO: You should modify this string to be something
    appropriate
// such as the name of your company or organization
SetRegistryKey(_T("Local␣AppWizard-Generated␣
    Applications"));
LoadStdProfileSettings(4);  // Load standard INI file
    options (including MRU)
// Register the application's document templates.
    Document templates
//  serve as the connection between documents, frame
    windows and views
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
     IDR_MAINFRAME,
     RUNTIME_CLASS(CTriangleDoc),
     RUNTIME_CLASS(CMainFrame),       // main SDI frame
         window
     RUNTIME_CLASS(CTriangleView));
if (!pDocTemplate)
     return FALSE;
AddDocTemplate(pDocTemplate);



// Parse command line for standard shell commands, DDE,
     file open
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
```

```
    // Dispatch commands specified on the command line.
       Will return FALSE if
    // app was launched with /RegServer, /Register, /
       Unregserver or /Unregister.
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so
       show and update it
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    // call DragAcceptFiles only if there's a suffix
    //  In an SDI app, this should occur after
       ProcessShellCommand
    return TRUE;
}



// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    enum { IDD = IDD_ABOUTBOX };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    //
       DDX/DDV support

// Implementation
protected:
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
```

```
END_MESSAGE_MAP()

// App command to run the dialog
void CTriangleApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}


// CTriangleApp message handlers
```

```
#pragma once


// CPrefDialog dialog

class CPrefDialog : public CDialog
{
    DECLARE_DYNAMIC(CPrefDialog)

public:
    CPrefDialog(CWnd* pParent = NULL);    // standard
        constructor
    virtual ~CPrefDialog();

// Dialog Data
    enum { IDD = IDD_PREFDIALOG };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);     //
        DDX/DDV support

    DECLARE_MESSAGE_MAP()
public:
    double m_Left;
    double m_Top;
    double m_Right;
    double m_Bottom;
    double m_Area;
    int m_bAnisotropic;
    int m_bLabels;
};
```

```
// PrefDialog.cpp : implementation file
//

#include "stdafx.h"
#include "MFCtriangle.h"
#include "PrefDialog.h"


// CPrefDialog dialog

IMPLEMENT_DYNAMIC(CPrefDialog, CDialog)

CPrefDialog::CPrefDialog(CWnd* pParent /*=NULL*/)
    : CDialog(CPrefDialog::IDD, pParent)
    , m_Left(0)
    , m_Top(100)
    , m_Right(100)
    , m_Bottom(0)
    , m_Area(1000)
    , m_bAnisotropic(false)
    , m_bLabels(false)
{

}

CPrefDialog::~CPrefDialog()
{
}

void CPrefDialog::DoDataExchange(CDataExchange* pDX)
{
    DDX_Text(pDX, IDC_eLeft, m_Left);
    DDX_Text(pDX, IDC_eTop, m_Top);
    DDX_Text(pDX, IDC_eRight, m_Right);
    DDX_Text(pDX, IDC_eBottom, m_Bottom);
    DDX_Text(pDX, IDC_eArea, m_Area);
    DDV_MinMaxDouble(pDX, m_Area, 10e-29, 10e300);
    DDX_Check(pDX, IDC_chAnisotropic, m_bAnisotropic);
    DDX_Check(pDX, IDC_chBoundaryLabels, m_bLabels);
}


BEGIN_MESSAGE_MAP(CPrefDialog, CDialog)
END_MESSAGE_MAP()


// CPrefDialog message handlers
```

```
#ifndef gRIGID
#define gRIGID
#include <iostream>
#include <cmath>
using namespace std;
namespace gRigid
{
struct row{int N;double*A;row(int n=0):N(n){if(n)A=new
    double[n];else A=0;}};
class Graph
{
    class Node
    {
    public:
        short Deg;  //degree
        short* Adj; //adjustant nodes
        Node():Deg(0),Adj(0){}
        Node(short deg,short* adj):Deg(deg),Adj(adj){}
    };
    class Level//set representation could be more effective
         if it was a linked list
    {
        short N;
    public:
        bool* Items;
        Level(int n):N(n){Items=new bool[N];memset(Items,0,
           N*sizeof(bool));}
        void Print(int no)
        {
            cout<<no<<":␣";
            for(int i=0;i<N;i++)
            if(Items[i])
                cout<<i<<"␣";
            cout<<endl;
        }
        bool operator-=(Level& l)
        {
            if(l.N!=N)
                throw int();
            bool isempty=true;
            for(int i=0;i<N;i++)
                if(Items[i])
                    if(l.Items[i])
                        Items[i]=0;
                    else
                        isempty=false;
            return isempty;
        }
        friend class Graph;
    };
    int N;  //number of nodes
    Node* Nodes;
    row* Env;
    int State;
    int* rcm;
    int* ircm;
public:
    static Graph* pCurrentGraph;
    //                           (                 ):
    //a11
    //a21 a22
    //a31 a32 a33
    //...
    //an1 ... ann
    Graph(int n,double**a):N(n),State(0)
    {
        pCurrentGraph=this;//need this for compare function
        Nodes=new Node[N];
        //each nodes' degree
        short* deg=new short[N];
        memset(deg,0,n*sizeof(short));
        for(int i=N-1;i>=0;i--)
            for(int j=i-1;j>=0;j--)
                if(a[i][j])
                {
                    deg[i]++;
                    deg[j]++;
                }
        //create graph
        for(int i=0;i<N;i++)
        {
            short* adj=new short[deg[i]];
            int ai=0;
            for(int j=0;j<i;j++)
                if(a[i][j])
                    adj[ai++]=j;
            for(int j=i+1;j<n;j++)
                if(a[j][i])
                    adj[ai++]=j;
            Nodes[i]=Node(deg[i],adj);
        }
        delete[] deg;
        //reverse Cuthill-McKee rearrangement
        rcm=RCM();
        //inverse reverse Cuthill-McKee
        ircm=new int[N];
        for(int i=0;i<N;i++)
            ircm[rcm[i]]=i;
        //allocate memory for Env(a)
        row* xenv=FindEnvelope(rcm,ircm);
        //now fill it with appropriate values
        for(int i=0;i<N;i++)
        {
            for(int j=0;j<xenv[i].N;j++)
            {
                int ii=rcm[i-j];
                int jj=rcm[i];
                if(ii<jj)
                {
                    int t=ii;
                    ii=jj;
                    jj=t;
                }
                xenv[i].A[xenv[i].N-j-1]=a[ii][jj];//cout
                    <<(xenv[i].A[xenv[i].N-j-1]=a[ii][jj])
                    <<" ";
            }//cout<<endl;
        }
        Env=xenv;
    }
    double* Solve(const double*b)
    {
        double* rhs=new double[N];
        for(int i=0;i<N;i++)
            rhs[i]=b[rcm[i]];
        if(State==0)//check if matrix has been factorized
        {
            EnvFact(Env);
            State=1;
        }
        EnvLowSolve(N,Env,rhs);
        EnvUpSolve(N,Env,rhs);
        double* res=new double[N];
        for(int i=0;i<N;i++)
            res[i]=rhs[ircm[i]];
        delete[] rhs;
        return res;
    }
    void Print()
    {
        for(int i=0;i<N;i++)
        {
            cout<<i<<":␣";
            for(int j=0;j<Nodes[i].Deg;j++)
                cout<<Nodes[i].Adj[j]<<"␣";
            cout<<endl;
        }
    }
    ~Graph()
    {
        for(int i=0;i<N;i++)
            delete[] Nodes[i].Adj;
        delete[] Nodes;
        delete[] rcm;
        delete[] ircm;
    }
private:
    Level Adj(Level l)//returns set adjustant to l
    {
        Level r(N);
        for(int i=0;i<N;i++)
            if(l.Items[i])
                for(int j=0;j<Nodes[i].Deg;j++)
                    r.Items[Nodes[i].Adj[j]]=1;
        return r;
    }
    //                (Gibbs)
                             (                              )

    //  1 (              ):                           r
    //  2 (                           ):
                                r:
    //L(r)={L[0](r),L[1](r),...,L[l(r)](r)}, l(r)-
                                r
    //  3 (                              ):           L[l(r)](
       r)
    //  4 (                           ):
                                :
    //L(x)={L[0](x),L[1](x),...,L[l(x)](r)}
    //    l(x)>l(r)               r:=x                      2
    //  5 (      ):x-
    //                                : L[0](x)={x},L[1](x)=Adj(x
       ),L[i](x)=Adj(L[i-1](x))-L[i-2](x),i=2..l(x)
    int PseudoPeripheral(int start=0)//pseudo peripheral
       root
    {
        int root=-1;
        int lev=start;//

        Level l=rlsLast(lev);//
        while(1)
        {
            //
            int deg=1000000;
            root=-1;
            for(int i=0;i<N;i++)
            if(l.Items[i])
                if(Nodes[i].Deg<deg)
                {
                    root=i;
                    deg=Nodes[i].Deg;
                };
        //
        int lev1=root;
        Level l1=rlsLast(lev1);
        if(lev1<=lev)
        {
            delete[] l.Items;
            delete[] l1.Items;
            break;
        }
        delete[] l.Items;
        l=l1;
        lev=lev1;
    }
    return root;
}
//            L[l(x)](x)
//                 L[l(x)](x),         x:=l(x)
Level rlsLast(int& x)//rooted level structure last
   element
{
    Level L0(N);
    L0.Items[x]=1;//L0.Print(0);
    Level L1=Adj(L0);//L1.Print(1);
    int i=2;
    while(1)
    {
        Level L2=Adj(L1);
        if(L2==L0)//returns true if the result is an
           empty set
        {
            delete[] L2.Items;
            break;
        }
        delete[] L0.Items;
        //L2.Print(i);
        i++;
        L0=L1;
        L1=L2;
    }
    delete[] L0.Items;
    x=i-1;
    return L1;
}
```

```
//                                    –
                                                            ,

//     1 :                                        r     x[0]:=r
//     2 (                  ):        i=0..N–1
                                    x[i]

//     3 (                                    ):y[i]=x[N–i–1],  i
    =0..N–1
//             gA              ,

int* RCM()//reverse Cuthill–McKee
{
     int r=PseudoPeripheral();
     int* queue=new int[N];
     queue[0]=r;
     int qi=0;
     int cqi=0;
     Level mask(N);mask.Items[r]=1;//
     while(qi<N–1)
     {
          if(cqi>qi)//
          {
               int s=–1;
               while(mask.Items[++s]);//

               queue[cqi]=PseudoPeripheral(s);
          }
          int c=queue[cqi];
          qsort(Nodes[c].Adj,Nodes[c].Deg,sizeof(short),
              compare);//

          for(int i=0;i<Nodes[c].Deg;i++)
               if(mask.Items[Nodes[c].Adj[i]]==0)//
                                  ,

               {
                    mask.Items[Nodes[c].Adj[i]]=1;
                    queue[++qi]=Nodes[c].Adj[i];
               }
               cqi++;//

     }
     delete[] mask.Items;
     int* revqueue=new int[N];
     for(int i=0;i<N;i++)
          revqueue[i]=queue[N–i–1];
     delete[] queue;
     return revqueue;
    }
    friend int compare(const void* elem1,const void* elem2)
        ;//
    row* FindEnvelope(int*perm,int*invp)//finds matrix
        structure and allocates memory for it
    {
        int bandw=0;
        int envsize=0;
        int envi=0;
        row* x=new row[N];
        //cout<<"env size:\n";
        for(int i=0;i<N;i++)
        {
            int iperm=perm[i];
            int ifirst=i;
            for(int j=0;j<Nodes[iperm].Deg;j++)
            {
                int nabor=Nodes[iperm].Adj[j];
                nabor=invp[nabor];
                if(nabor<ifirst)
                    ifirst=nabor;
            }
            envi=i-ifirst;
            if(envi>bandw)
                bandw=envi;
            envsize+=envi;
            x[i]=row(envi+1);
            //cout<<envi<<" ";
        }
#ifdef gDEBUG
        cerr<<"Env=\t"<<envsize<<endl;
#endif
        //cout<<endl;
        return x;
    }
    double* EnvLowSolve(int n,row* a,double*b)
    {
        //ifirst has the index of the first non-zero right
            hand vector
        int ifirst=0;for(;ifirst<n;ifirst++)
            if(b[ifirst])
                break;
        //last holds the index of last computed nonzero
            result array element
        int last=-1;
        for(int i=ifirst;i<n;i++)
        {
            int iband=a[i].N;
            int di=a[i].N-1;//diagonal index
            double* cr=a[i].A;//current row
            double s=b[i];
            if(iband>i+1)iband=i+1;
            int l=i-iband+1;
            b[i]=0;
            if((iband!=1)&&(last>=l))
            {
                for(int k=a[i].N-iband;k<di;k++)
                    s-=cr[k]*b[l++];
            }
            if(s)
            {
                b[i]=s/cr[di];
                last=i;
            }
        }
        return b;
    }
    double* EnvUpSolve(int n,row* a,double*b)
    {
        int i=n;
        while(--i)
        {
            if(b[i]==0)
                continue;
            double* cr=a[i].A;
            double s=b[i]/cr[a[i].N-1];
            b[i]=s;
            int iband=a[i].N;
            if(iband>i+1)
                iband=i+1;
            if(iband==1)
                continue;
            int l=a[i].N-2;
            for(int k=i-1;k>=i-iband+1;k--)
                b[k]-=s*cr[l--];
        }
        b[0]/=a[i].A[0];
        return b;
    }
    row* EnvFact(row* a)
    {
        if(a[0].A[0]<=0)
            throw int();//not positively definate matrix
        a[0].A[0]=sqrt(a[0].A[0]);
        for(int i=1;i<N;i++)
        {
            double* cr=a[i].A;
            int iband=a[i].N-1;
            double temp=cr[iband];
            if(iband>0)
            {
                int ifirst=i-iband;
                EnvLowSolve(iband,a+ifirst,cr);
                for(int j=0;j<iband;j++)
                    temp-=cr[j]*cr[j];
            }
            if(temp<=0)
                throw int();//not positively definate
                   matrix
            cr[iband]=sqrt(temp);
        }
        return a;
    }
};
//Graph* Graph::pCurrentGraph=0;
int compare(const void* elem1,const void* elem2);
/*{
    short e1=*((short*)elem1);
    short e2=*((short*)elem2);
    if(Graph::pCurrentGraph->Nodes[e1].Deg>Graph::
       pCurrentGraph->Nodes[e2].Deg)
        return 1;
    else
        return -1;
}*/
int Env(int n,double**a);
/*{
    int size=0;
    for(int i=0;i<n;i++)
    {
        int j=0;
        size+=i;
        for(;(a[i][j]==0)&&(i-j>0);j++)
            size--;
    }
    return size;
}*/
}
#endif
```

```
#include "stdafx.h"
#include "RIGID.H"


namespace gRigid
{
Graph* Graph::pCurrentGraph=0;
int compare(const void* elem1,const void* elem2)
{
     short e1=*((short*)elem1);
     short e2=*((short*)elem2);
     if(Graph::pCurrentGraph->Nodes[e1].Deg>Graph::
         pCurrentGraph->Nodes[e2].Deg)
          return 1;
     else
          return -1;
}
int Env(int n,double**a)
{
     int size=0;
     for(int i=0;i<n;i++)
     {
          int j=0;
          size+=i;
          for(;(a[i][j]==0)&&(i-j>0);j++)
               size--;
     }
     return size;
}
}
```

```
#pragma once
#include "afxwin.h"
#include "afxcmn.h"
#include "TriangleDoc.h"


// CSolveDialog dialog

class CSolveDialog : public CDialog
{
    DECLARE_DYNAMIC(CSolveDialog)
    gPOINT minMax;
public:
    CSolveDialog(CWnd* pParent = NULL);   // standard
        constructor
    virtual ~CSolveDialog();

// Dialog Data
    enum { IDD = IDD_SOLVEDIALOG };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    //
        DDX/DDV support
    virtual BOOL OnInitDialog();
public:
    double m_dt;
    int m_kt;
    double m_x;
    double m_y;
    double m_t;
    CEdit m_editResult;
    CEdit m_editStep;
    CEdit m_editT0;
    CEdit m_editkt;
    CSliderCtrl m_sliderStep;
protected:
    afx_msg void OnBnClickedbsolve();
    afx_msg void OnDeltaposspinstep(NMHDR *pNMHDR, LRESULT
        *pResult);
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos,
        CScrollBar* pScrollBar);
    afx_msg void OnBnClickedbeval();
    DECLARE_MESSAGE_MAP()
};
```

```
// SolveDialog.cpp : implementation file
//

#include "stdafx.h"
#include "MFCtriangle.h"
#include "SolveDialog.h"
#include "TriangleDoc.h"
#include "MainFrm.h"
#include "RIGID.h"
#include "TriangleView.h"
#include "fparser.h"


// CSolveDialog dialog

IMPLEMENT_DYNAMIC(CSolveDialog, CDialog)

CSolveDialog::CSolveDialog(CWnd* pParent /*=NULL*/)
    : CDialog(CSolveDialog::IDD, pParent)
    , m_dt(20)
    , m_kt(30)
    , m_x(0)
    , m_y(0)
    , m_t(0)
{
}

CSolveDialog::~CSolveDialog()
{
}

void CSolveDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Text(pDX, IDC_edt, m_dt);
    DDV_MinMaxDouble(pDX, m_dt, 0, 1000000);
    DDX_Text(pDX, IDC_ekt, m_kt);
    //DDV_MinMaxInt(pDX, m_kt, 1, 500);
    DDX_Text(pDX, IDC_EvX, m_x);
    DDX_Text(pDX, IDC_EvY, m_y);
    DDX_Text(pDX, IDC_EvT, m_t);
    DDV_MinMaxDouble(pDX, m_t, 0, 1000000);
    DDX_Control(pDX, IDC_eResult, m_editResult);
    DDX_Control(pDX, IDC_eStep, m_editStep);
    DDX_Control(pDX, IDC_eT0, m_editT0);
    DDX_Control(pDX, IDC_ekt, m_editkt);
    DDX_Control(pDX, IDC_sliderStep, m_sliderStep);
}
```

```
BEGIN_MESSAGE_MAP(CSolveDialog, CDialog)
    ON_BN_CLICKED(IDC_bSolve, &CSolveDialog::
        OnBnClickedbsolve)
    ON_NOTIFY(UDN_DELTAPOS, IDC_spinStep, &CSolveDialog::
        OnDeltaposspinstep)
    ON_WM_HSCROLL()
    ON_BN_CLICKED(IDC_bEval, &CSolveDialog::
        OnBnClickedbeval)
END_MESSAGE_MAP()


// CSolveDialog message handlers

BOOL CSolveDialog::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_editT0.SetWindowText(_T("0"));
    m_editStep.SetWindowText(_T("0"));
    return TRUE;  // return TRUE unless you set the focus
        to a control
}

//multiplies matrix A by vector b,assuming A is lower
    symetric
void MxV(int size,double**A,double*b,double*r)
{
    for(int i=0;i<size;i++)
    {
        r[i]=0;
        int j=0;for(;j<i;j++)
            r[i]+=A[i][j]*b[j];
        for(j=i;j<size;j++)
            r[i]+=A[j][i]*b[j];
    }
}
char* conv(const wchar_t* s);
double ustart(double x,double y)
{
    if( x>0.6 && x<0.7 && y>0.4 && y<0.5)
        return 1;
    else
        return 0;
}
void CSolveDialog::OnBnClickedbsolve()
{
    UpdateData(TRUE);
    CMainFrame* frame=(CMainFrame*)AfxGetMainWnd();
    CTriangleDoc* pDoc=(CTriangleDoc*)frame->
```

```
        GetActiveDocument();
    ASSERT(pDoc->IsKindOf(RUNTIME_CLASS(CTriangleDoc)));
    vector<double*>& U=pDoc->U;
    for(int i=0;i<U.size();i++)
        delete[] U[i];
    U.clear();
//...allocate memory
    int n=pDoc->m_ptVector.size();
    if(n<3)
    {
        MessageBox(_T("No␣geometry␣defined!"),_T("error!"))
            ;
        return;
    }
    //for heat with p2 or isoparametric p2
    if(CTriangle::type!=CTriangle::linear)
        n=n+(3*pDoc->m_triList.size()+pDoc->m_segList.size
            ())/2;

    double** K=new double*[n];
    double** M=new double*[n];
    for(int i=0;i<n;i++)
    {
        K[i]=new double[n];
        memset(K[i],0,n*sizeof(double));
        M[i]=new double[n];
        memset(M[i],0,n*sizeof(double));
    }
    double* Q=new double[n];
    memset(Q,0,n*sizeof(double));
    //let CTriangle know location of points
    CTriangle::v=pDoc->m_ptVector;
    CTriangle::splines=pDoc->domain.splines;
//...assemble matrices
    for(list<CTriangle>::iterator i=pDoc->m_triList.begin()
        ;i!=pDoc->m_triList.end();i++)
    {
        (*i).Contribute(K,Q);
        (*i).ContributeMass(M);
    }
    //accommodate for boundaries
    /*gBoundary::Init(K,Q,pDoc->m_ptVector);
    for(list<CSegment>::iterator i=pDoc->m_segList.begin();
        i!=pDoc->m_segList.end();i++)
        pDoc->m_bndVector[i->boundary]->Accomodate(i);//
            this function is virtual: either Dirichlet or
            Newton*/
//...compute U[0]
    double* u0=new double[n];
    //evaluate function from string to u0 in points here...
    memset(u0,0,n*sizeof(double));
    fparser::FunctionParser parser;
    wchar_t buffer[100];
    m_editT0.GetWindowTextW(buffer,100);
    std::string T0(conv(buffer));
    int e=parser.Parse(T0,"x,y");
    parser.Optimize();
    double vars[2];
    for(int i=0;i<pDoc->m_ptVector.size();++i)
    {
        vars[0]=pDoc->m_ptVector[i].x;
        vars[1]=pDoc->m_ptVector[i].y;
        u0[i]=ustart(pDoc->m_ptVector[i].x,pDoc->m_ptVector
            [i].y);
        //u0[i]=parser.Eval(vars);
    }
//
    U.push_back(u0);

    for(int i=0;i<n;++i)
        for(int j=0;j<n;++j)
            M[i][j]+=0.5*m_dt*K[i][j];
    gRigid::Graph g(n,M);
    double* Kak=new double[n];
    for(int k=0;k<m_kt;++k)
    {
        for(int i=0;i<n;++i)
        {
            Kak[i]=0;
            for(int j=0;j<n;++j)
                Kak[i]+=K[i][j]*U[k][j];
        }
        //MxV(n,K,U[k],Kak);
        for(int i=0;i<n;++i)
            Kak[i]-=Q[i];
        double* alpha=g.Solve(Kak);
        for(int i=0;i<n;++i)
            alpha[i]=U[k][i]-m_dt*alpha[i];
        for(list<CSegment>::iterator i=pDoc->m_segList.
            begin();i!=pDoc->m_segList.end();i++)
            if(i->boundary==0)
                alpha[i->x]=alpha[i->y]=0;
        U.push_back(alpha);
    }

    /*int* Indy=new int[n];for(int i=0;i<n;i++)Indy[i]=i;

    for(list<CSegment>::iterator i=pDoc->m_segList.begin();
        i!=pDoc->m_segList.end();i++)
         if(i->boundary==0)
         {
             Indy[i->x]=-1;
             Indy[i->y]=-1;
         }
    int Ind_n=0;for(int i=0;i<n;i++)if(Indy[i]!=-1)Ind_n++;

    int* Int=new int[Ind_n];int Int_i=0;
    for(int i=0;i<n;i++)
         if(Indy[i]!=-1)Int[Int_i++]=Indy[i];
    //setup M_Ind
    double** M_Ind=new double*[Ind_n];
    for(int i=0;i<Ind_n;i++)
    {
        M_Ind[i]=new double[Ind_n];
        for(int j=0;j<Ind_n;j++)
            M_Ind[i][j]=M[Int[i]][Int[j]];
    }


    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            M[i][j]-=m_dt*K[i][j];
    gRigid::Graph g(Ind_n,M_Ind);

    double* Kak=new double[n];
    double* Kak_Ind=new double[Ind_n];

    for(int k=0;k<m_kt;++k)
    {
        for(int i=0;i<n;++i)
        {
            Kak[i]=0;
            for(int j=0;j<n;++j)
                Kak[i]+=M[i][j]*U[k][j];
        }
        //MxV(n,K,U[k],Kak);
        for(int i=0;i<n;++i)
            Kak[i]+=Q[i];
        for(int i=0;i<Ind_n;i++)
            Kak_Ind[i]=Kak[Int[i]];


        double* alpha=g.Solve(Kak_Ind);


        double* beta=new double[n];
        for(int i=0;i<n;++i)
```

```
                beta[i]=U[k][i];
            for(int i=0;i<Ind_n;i++)
                beta[Int[i]]=alpha[i];
            //set dirichlet values
            U.push_back(beta);
        }*/
//...free memory
        for(int i=0;i<n;++i)
        {
            delete[]K[i];
            delete[]M[i];
        }
        delete[]K;
        delete[]M;
        delete[]Q;
//...output
        pDoc->u=U[0];
        m_sliderStep.SetRange(0,m_kt-1,TRUE);
        pDoc->m_bSolved=true;
        gPOINT res(pDoc->u[0],pDoc->u[0]);
/*      for(int j=0;j<U.size();++j)
            for(int i=1;i<n;i++)
            if(pDoc->U[j][i]>res.x)
                res.x=pDoc->U[j][i];
            else
                if(pDoc->U[j][i]<res.y)
                    res.y=pDoc->U[j][i];*/
        for(int i=1;i<n;i++)
            if(pDoc->u[i]>res.x)
                res.x=pDoc->u[i];
            else
                if(pDoc->u[i]<res.y)
                    res.y=pDoc->u[i];
        CTriangleView* view=(CTriangleView*)frame->
            m_wndSplitter.GetPane(0,1);
        ASSERT(view->IsKindOf(RUNTIME_CLASS(CTriangleView)));
        minMax=view->minmax=res;
        view->Invalidate(FALSE);
        pDoc->m_dt=m_dt;
        pDoc->m_bSolved=true;
pDoc->UpdateAllViews(0,1,(CObject*)&res);
}

void CSolveDialog::OnDeltaposspinstep(NMHDR *pNMHDR,
    LRESULT *pResult)
{
        LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(
            pNMHDR);
        *pResult = 0;
        int newpos=pNMUpDown->iPos+pNMUpDown->iDelta;
        if((newpos<0)||(newpos>=m_kt))
        {
            *pResult=1;
            return;
        }
        CMainFrame* frame=(CMainFrame*)AfxGetMainWnd();
        CTriangleDoc* pDoc=(CTriangleDoc*)frame->
            GetActiveDocument();
        ASSERT(pDoc->IsKindOf(RUNTIME_CLASS(CTriangleDoc)));
        pDoc->u=pDoc->U[newpos];
        gPOINT res(pDoc->u[0],pDoc->u[0]);
        int n=pDoc->m_ptVector.size();
        for(int i=1;i<n;i++)
            if(pDoc->u[i]>res.x)
                res.x=pDoc->u[i];
            else
                if(pDoc->u[i]<res.y)
                    res.y=pDoc->u[i];
        //pDoc->UpdateAllViews(0,1,(CObject*)&res);
        CTriangleView* view=(CTriangleView*)frame->
            m_wndSplitter.GetPane(0,1);
        ASSERT(view->IsKindOf(RUNTIME_CLASS(CTriangleView)));
        view->minmax=res;
        view->Invalidate(FALSE);
}


void CSolveDialog::OnHScroll(UINT nSBCode, UINT nPos,
        CScrollBar* pScrollBar)
{
        CSliderCtrl* slider=(CSliderCtrl*)pScrollBar;
        int pos=slider->GetPos();
        CMainFrame* frame=(CMainFrame*)AfxGetMainWnd();
        CTriangleDoc* pDoc=(CTriangleDoc*)frame->
            GetActiveDocument();
        ASSERT(pDoc->IsKindOf(RUNTIME_CLASS(CTriangleDoc)));
        pDoc->u=pDoc->U[pos];
        gPOINT res(pDoc->u[0],pDoc->u[0]);
        int n=pDoc->m_ptVector.size();
        for(int i=1;i<n;i++)
            if(pDoc->u[i]>res.x)
                res.x=pDoc->u[i];
            else
                if(pDoc->u[i]<res.y)
                    res.y=pDoc->u[i];
        CTriangleView* view=(CTriangleView*)frame->
            m_wndSplitter.GetPane(0,1);
        ASSERT(view->IsKindOf(RUNTIME_CLASS(CTriangleView)));
        view->minmax=res;
        view->Invalidate(FALSE);
}

void CSolveDialog::OnBnClickedbeval()
{
        UpdateData(1);
        CString result;

        CMainFrame* frame=(CMainFrame*)AfxGetMainWnd();
        CTriangleDoc* pDoc=(CTriangleDoc*)frame->
            GetActiveDocument();
        ASSERT(pDoc->IsKindOf(RUNTIME_CLASS(CTriangleDoc)));

        double r=pDoc->Eval(m_x,m_y,m_t);
        result.Format(_T("%f"),r);
        m_editResult.SetWindowText(result);
}
```

```
#pragma once
#include <list>
#include <vector>
#include <algorithm>
#include <iostream>
#include "triangle.h"
#include "rigid.h"
#include "fparser.h"
//...this class is used when we want to generate a domain
//   based on a list of points and normals
struct PointAndNormal
{
    double x,y;
    double nx,ny;
public:
    PointAndNormal(){}
    PointAndNormal(double X,double Y,double nX,double nY):x
       (X),y(Y),nx(nX),ny(nY){}
};
//...this is an interface to problem:
//   dT/dt=  (kT)+f
//
//   k11,k12,k22,f are used for integration
//   K11,K12,K22,F are used if we want to change problem in
    runtime
//   in this program CSubdomainDialog does this
namespace HeatConductionProblem
{
    extern fparser::FunctionParser K11,K12,K22,F;
    extern double (*k11)(double x,double y);
    extern double (*k22)(double x,double y);
    extern double (*k12)(double x,double y);
    extern double (*f)(double x,double y);
};
//////////////////////////////
struct gPOINT
{
    double x,y;
    struct aux
    {
        int border;          //...what kind of boundary
           condition, 0 means 0-Dirichlet
        int spline_index1;  //..it's part of a spline if it
           's not -1
        double t1;          //...spline coordinate
        int spline_index2;  //..suppose that spline_index1
           is in direction of list,
                            //..then spline_index2 is in
                               other direction
        double t2;           //...spline coordinate along
            the other spline
    public:
        aux(int b=0,int s_i1=-1,double T1=0,int s_i2=-1,
            double T2=0):border(b),
             spline_index1(s_i1),spline_index2(s_i2),t1(T1),
                t2(T2){}
    }* misc;
public:
    gPOINT():misc(0){}
    gPOINT(double X,double Y):x(X),y(Y),misc(0){}
    gPOINT(double X,double Y,int b):x(X),y(Y){misc=new aux(
        b);}
    gPOINT(CPoint p):x(p.x),y(p.y){}
    gPOINT(double X,double Y,int b,int spline_index1,double
         t1,int spline_index2=-1,double t2=0):x(X),y(Y)
    {misc=new aux(b,spline_index1,t1,spline_index2,t2);}
    //...operations
    inline gPOINT operator *(double a){return gPOINT(x*a,y*
        a);}
    inline friend gPOINT operator +(gPOINT a,gPOINT b){
        return gPOINT(a.x+b.x,a.y+b.y);}
    //...attributes
    inline operator POINT() const{return CPoint(int(x),int(y
        ));}
    inline bool curved() const{if(misc) if(misc->
        spline_index1!=-1)return true;return false;}
    inline int border() const{if(misc)return misc->border;
        else return 0;}
    inline void setBorder(int b){if(misc)misc->border=b;
        else misc=new aux(b);}
    friend inline double length(gPOINT a,gPOINT b){return
        sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));}
};
struct Spline
{
    gPOINT s;
    gPOINT m;
    gPOINT t;
public:
    Spline(){}
    Spline(gPOINT S,gPOINT M,gPOINT T):s(S),m(M),t(T){}

    gPOINT at(double k)
    {
        return gPOINT(k*k*s.x+2*k*(1-k)*m.x+(1-k)*(1-k)*t.x
            ,
                    k*k*s.y+2*k*(1-k)*m.y+(1-k)*(1-k)*t.y
                        ,s.border());//?,k
    }
};
class CTriangleView;
struct Domain
{
    list<gPOINT> points;
    vector<Spline> splines;

    typedef list<gPOINT>::iterator li;
    li currVertex;
    void AddSpline(gPOINT midpoint, li* at=0);
public:
    Domain(){};
    void clear();
    void Draw(CDC* pDC, CTriangleView* view);
    //select vertex close to this point
    void SelectVertexAt(gPOINT at);
    //select vertex corresponding to closest curve
    void SelectCurveAt(gPOINT at);
    //add or move control point of the spline
    void SetMidpoint(gPOINT at);
    void AddVertex(gPOINT v);
    //all following operations are on current vertex
    void MoveVertex(gPOINT v);
    void DeleteVertex();
    void NextVertex();
    void PrevVertex();
    void ToggleCurved();
    bool GetCurved();
    inline void SetBorder(int b){currVertex->setBorder(b);}
    inline int  GetBorder(){return currVertex->border();}
    //length of current spline
    double CurveLength(li i);
};
//////////////////////////////
struct CSegment
{
    //...indices of start and end points of segment
    int x,y;
    //...index of midpoint of segment
    int m;
    //...index of boundary condition
    int boundary;
    CSegment(int i1,int i2,int bound):x(i1),y(i2),boundary(
       bound){}
    CSegment(int i1,int i2,int i3,int bound):x(i1),y(i2),m(
       i3),boundary(bound){}
};
//////////////////////////////
```

```
struct vect
{
    double v[2];
public:
    vect(double v1,double v2)
    {
        v[0]=v1;
        v[1]=v2;
    }
    vect operator =(double a)
    {
        _ASSERTE(a==0);
        v[0]=v[1]=0;
        return *this;
    }
    double operator[](int i)
    {
        _ASSERTE( (i==0)||(i==1) );
        if(i==0)
            return v[0];
        else
            return v[1];
    }
    friend double operator*(vect a,vect b)
    {
        return a.v[0]*b.v[0]+a.v[1]*b.v[1];
    }
};
typedef double (*fptr)(double,double);
typedef vect  (*fptrv)(double,double);
struct matr
{
    double m[2][2];
public:
    matr(double m11,double m12,double m21,double m22)
    {
        m[0][0]=m11;m[0][1]=m12;
        m[1][0]=m21;m[1][1]=m22;
    }
    friend vect operator *(matr a,vect b)
    {
        return vect(a.m[0][0]*b[0]+a.m[0][1]*b[1],a.m
            [1][0]*b[0]+a.m[1][1]*b[1]);
    }
    inline friend double det(matr m)
    {
        return m.m[0][0]*m.m[1][1]-m.m[0][1]*m.m[1][0];
    }
    friend matr inv(matr m)
    {
        double detm=det(m);
        return matr(m.m[1][1]/detm,-m.m[0][1]/detm,-m.m
            [1][0]/detm,m.m[0][0]/detm);
    }
    friend matr invt(matr m)
    {
        double detm=det(m);
        return matr(m.m[1][1]/detm,-m.m[1][0]/detm,-m.m
            [0][1]/detm,m.m[0][0]/detm);
    }
};

struct CTriangle
{
    enum approximation_type{linear,quadratic,isoparametric
        };
    static approximation_type type;
    enum quadrature_type{gauss2,gauss3};
    static quadrature_type qtype;
    //...indices of corner points in global matrix
    int i1,i2,i3;
    //...indices of middle points in global matrix
    int m1,m2,m3;
    static inline double tarea(gPOINT a,gPOINT b,gPOINT c){
        return fabs((b.x*c.y-b.y*c.x)+(a.y*c.x-a.x*c.y)+(a.x
        *b.y-a.y*b.x))/2;}
    //...parameters of isoparametric transformation
    double dx,dy;
public:
    //...address of points of triangulation
    static vector<gPOINT>& v;
    static vector<Spline>& splines;
    CTriangle(int I1,int I2,int I3):i1(I1),i2(I2),i3(I3){}
    CTriangle(int I1,int I2,int I3,int M1,int M2,int M3):i1
        (I1),i2(I2),i3(I3),m1(M1),m2(M2),m3(M3){}
    CTriangle(){};
    //...operations

    //...(xFk,yFk) is transformation from reference
        triangle to real triangle
    double xFk(double x,double y)
    {
        double x1=v[i1].x,x2=v[i2].x,x3=v[i3].x;
        return x1*(1-x-y)+x2*(x)+x3*(y) +4*x*y*dx;
    }
    double yFk(double x,double y)
    {
        double y1=v[i1].y,y2=v[i2].y,y3=v[i3].y;
        return y1*(1-x-y)+y2*(x)+y3*(y) +4*x*y*dy;
    }
    //...detBk is the jacobian of transformation Fk
    double detBk()
    {
        double x1=v[i1].x,x2=v[i2].x,x3=v[i3].x;
        double y1=v[i1].y,y2=v[i2].y,y3=v[i3].y;
        double Bk[2][2]={{x2-x1,x3-x1},
                         {y2-y1,y3-y1}};
        return Bk[0][0]*Bk[1][1] -Bk[0][1]*Bk[1][0];
    }
    //...this is like gradient of transformation Fk
    matr BK(double x,double y)
    {
        double x1=v[i1].x,x2=v[i2].x,x3=v[i3].x;
        double y1=v[i1].y,y2=v[i2].y,y3=v[i3].y;
        return matr(x2-x1+4*y*dx,x3-x1+4*y*dy,y2-y1+4*x*dx,
           y3-y1+4*x*dy);
    }
    //...integrate a function over triangle
    double int2(fptr f)
    {
        double x[]={0.211324865,0.211324865,
            0.788675134,0.788675134};
        double y[]={0.166666667,0.622008467,
            0.044658198,0.166666667};
        double c[]={0.197168783,0.197168783,
            0.052831216,0.052831216};
        double res=0;
        int n=4;
        for(int i=0;i<n;i++)
            res+=c[i]*f(xFk(x[i],y[i]),yFk(x[i],y[i]));
        return detBk()*res;
    }
    //...shape functions on a reference triangle and their
        gradients
    inline static double P1_N1(double x,double y){return 1-
        x-y;}
    inline static double P1_N2(double x,double y){return x
        ;}
    inline static double P1_N3(double x,double y){return y
        ;}
    static fptr P1_N[3];

    inline static vect  P1_gradN1(double x,double y){
        return vect(-1,-1);}
    inline static vect  P1_gradN2(double x,double y){
        return vect(1,0);}
    inline static vect  P1_gradN3(double x,double y){
        return vect(0,1);}
    static fptrv P1_gradN[3];

    inline static double P2_N1(double x,double y){return
        (1-x-y)*(1-2*x-2*y);}
    inline static double P2_N2(double x,double y){return x
        *(2*x-1);}
    inline static double P2_N3(double x,double y){return y
        *(2*y-1);}
    inline static double P2_N4(double x,double y){return 4*
        x*y;}
    inline static double P2_N5(double x,double y){return 4*
        y*(1-x-y);}
    inline static double P2_N6(double x,double y){return 4*
        x*(1-x-y);}
    static fptr P2_N[6];

    inline static vect   P2_gradN1(double x,double y){
        return vect(4*x+4*y-3,4*x+4*y-3);}
    inline static vect   P2_gradN2(double x,double y){
        return vect(4*x-1,0);}
    inline static vect   P2_gradN3(double x,double y){
        return vect(0,4*y-1);}
    inline static vect   P2_gradN4(double x,double y){
        return vect(4*y,4*x);}
    inline static vect   P2_gradN5(double x,double y){
        return vect(-4*y,4-8*y-4*x);}
    inline static vect   P2_gradN6(double x,double y){
        return vect(4-4*y-8*x,-4*x);}
    static fptrv P2_gradN[6];

    void Contribute(double** K,double* Q);
    void ContributeMass(double** M);
    //this computes isoparametric middle point
    gPOINT p4(gPOINT p1,gPOINT p2);
    double Eval(double* u,double x,double y)
    {
        //TODO: use Bezier splines for evaluation

        double area=tarea(v[i1],v[i2],v[i3]);
        return (u[i1]*tarea(gPOINT(x,y),v[i2],v[i3])+u[i2]*
           tarea(v[i1],gPOINT(x,y),v[i3])+u[i3]*tarea(v[i1
           ],v[i2],gPOINT(x,y)))/area;
    }
    double integral(double* u)
    {
        if(type==linear)//0,405284734569351
            return (u[i1]+u[i2]+u[i3])*tarea(v[i1],v[i2],v[
                i3])/3;
```

```
        else
            return (u[m1]+u[m2]+u[m3])*tarea(v[i1],v[i2],v[
                i3])/3;
    }
    //...attributes
    bool Holds(double x,double y)
    {
        const double eps=10e-10;
        double t1=tarea(v[i1],v[i2],v[i3]);
        double t2=tarea(gPOINT(x,y),v[i2],v[i3])+tarea(v[i1
            ],gPOINT(x,y),v[i3])+tarea(v[i1],v[i2],gPOINT(x,
            y));
        if(fabs(t1-t2)<eps)
            return true;
        else
            return false;
    }
};
//////////////////////////////
class gBoundary
{
protected:
    //...address of global stiffness matrix
    static double**K;
    //...address of global right hand side
    static double*Q;
    //...address of points of triangulation
    static vector<gPOINT>& points;
public:
    static void Init(double** _K,double* _Q,vector<gPOINT>&
        _points){K=_K,Q=_Q;points=_points;}
    //...accomodate for current segment
    virtual void Accomodate(list<CSegment>::iterator i)=0;
    //...text information on boundary type and parameters
    virtual CString Info(int component){return CString(_T("
        error"));}
    //...save to file
    virtual void Serialize(CArchive& ar)=0;
    virtual ~gBoundary(){};
};
//heat flux boundary:   ? (kT)=q0+h*(T_inf-T)
class HeatFluxBoundary:public gBoundary
{
public:
    double q0;
    double h;
    double T_inf;
    HeatFluxBoundary(double _q0,double _h,double _T_inf):q0
        (_q0),h(_h),T_inf(_T_inf){}
    //...have to change this
    virtual void Accomodate(list<CSegment>::iterator i)
    {
        int i1=(*i).x;
        int i2=(*i).y;
        double L=length(points[i1],points[i2]);
        double b_c=(h*T_inf+q0);
        K[i1][i1]+=h*L/3;
        K[i1][i2]+=h*L/6;
        K[i2][i1]+=h*L/6;
        K[i2][i2]+=h*L/3;
        Q[i1]+=b_c*0.5*L;
        Q[i2]+=b_c*0.5*L;
    }
    virtual CString Info(int component)
    {
        CString result;
        switch(component)
        {
        case 1:
            result.Format(_T("%g"),q0);
            return result;
        case 2:
            result.Format(_T("%g"),h);
            return result;
        case 3:
            result.Format(_T("%g"),T_inf);
            return result;
        default:
            return CString(_T("-"));
        }
    }
    virtual void Serialize(CArchive& ar)
    {
        if(ar.IsStoring())
        {
            int kind=1;
            ar<<kind<<q0<<h<<T_inf;
        }
        else
        {
            ar>>q0>>h>>T_inf;
        }
    }
};
//temperature boudnary: T=T0
class TemperatureBoundary:public gBoundary
{
    double T0;
```

```
public:
    TemperatureBoundary(double _T0):T0(_T0){}
    virtual void Accomodate(list<CSegment>::iterator i)
    {
        int i1=(*i).x;
        int i2=(*i).y;
        int i3=(*i).m;
        K[i1][i1]=10e20;
        K[i2][i2]=10e20;
        Q[i1]=10e20*T0;
        Q[i2]=10e20*T0;
        if(CTriangle::type!=CTriangle::linear)
        {
            K[i3][i3]=10e20;//middle point
            Q[i3]=10e20*T0;
        }
    }
    virtual CString Info(int component)
    {
        CString result;
        switch(component)
        {
        case 0:
            result.Format(_T("%g"),T0);
            return result;
        default:
            return CString(_T("-"));
        }
    }
    virtual void Serialize(CArchive& ar)
    {
        if(ar.IsStoring())
        {
            int kind=0;
            ar<<kind<<T0;
        }
        else
        {
            ar>>T0;
        }
    }

};
//////////////////////////////
class Triangulation
{
public:
    //static vector<Spline> splines;
    //list of points
    list<gPOINT> m_ptList;
    bool tri;
    Domain domain;

    //double CurveLength(list<gPOINT>::iterator i);
public:
    vector<gPOINT> points;
    vector<gPOINT> midpoints;
    list<CTriangle> triangles;
    list<CSegment> segments;

    Triangulation(list<gPOINT>* ptList=0):tri(false)
    {
        if(ptList)
            m_ptList=*ptList;
    }
    Triangulation(vector<PointAndNormal> p)
    {
    }
    Triangulation(Domain* d)
    {
        domain=*d;
    }

    void triangulate(double maxarea);
    void refine(){}
    void clear()
    {
        points.clear();
        midpoints.clear();
        triangles.clear();
        segments.clear();
    }
};
struct problem
{
    vector<gBoundary*> bndVector;
    Triangulation T;
    double* u;
public:
    double Eval(gPOINT p)
    {
        list<CTriangle>::iterator i=T.triangles.begin();
        for(;i!=T.triangles.end();i++)
            if((*i).Holds(p.x,p.y))
                break;
        if(i==T.triangles.end())
            return 10e20;
        return (*i).Eval(u,p.x,p.y);
    }
    problem(){bndVector.push_back(new TemperatureBoundary
        (0));}
    void Solve()
    {
        delete[]u;
        //allocate memory
        int n=T.points.size();//number of dof depends on
            type of approximation P1 or P2
        double** K=new double*[n];
        for(int i=0;i<n;i++)
        {
            K[i]=new double[n];
            memset(K[i],0,n*sizeof(double));
        }
        double* Q=new double[n];
        memset(Q,0,n*sizeof(double));
        CTriangle::v=T.points;
        CTriangle::splines=T.domain.splines;
        //assemble matrices
        for(list<CTriangle>::iterator i=T.triangles.begin()
            ;i!=T.triangles.end();i++)
            (*i).Contribute(K,Q);
        //accommodate for boundaries
        gBoundary::Init(K,Q,T.points);
        for(list<CSegment>::iterator i=T.segments.begin();i
            !=T.segments.end();i++)
            bndVector[i->boundary-10]->Accomodate(i);//this
                function is virtual: either Dirichlet or
               Newton
        //solve linear system
        gRigid::Graph g(n,K);
        u=g.Solve(Q);
        //free memory
        delete[]Q;
        for(int i=0;i<n;i++)
            delete[] K[i];
        delete[]K;
    }
};
```

```
#include "stdafx.h"
#include "stage.h"
#include "TriangleView.h"

namespace HeatConductionProblem
{
    fparser::FunctionParser K11,K12,K22,F;
    double _k11(double x,double y)
    {
        double vars[2]={x,y};
        return K11.Eval(vars);
    }
    double __k11(double x,double y){return 1;}
    double _k22(double x,double y)
    {
        double vars[2]={x,y};
        return K22.Eval(vars);
    }
    double __k22(double x,double y){return 1;}
    double _k12(double x,double y)
    {
        double vars[2]={x,y};
        return K12.Eval(vars);
    }
    double __k12(double x,double y){return 0;}
    double _f(double x,double y)
    {
        double vars[2]={x,y};
        return F.Eval(vars);
    }
    double __f(double x,double y)
    {
        double pi=3.14159265358979323846264338327950;
        return 2*pi*pi*sin(pi*x)*sin(pi*y);
        return 1;
    }
    const bool dynamic=true;
    fptr k11=dynamic?_k11:__k11,
         k12=dynamic?_k12:__k12,
         k22=dynamic?_k22:__k22,
         f=  dynamic?_f  :__f;
};

gPOINT prev(list<gPOINT>& l,list<gPOINT>::iterator j)
{
    list<gPOINT>::iterator i=j;
    if(i==l.begin())
        return l.back();
    else
```

```
        return *--i;
}
gPOINT next(list<gPOINT>& l,const list<gPOINT>::iterator j)
{
    list<gPOINT>::iterator i=j;
    if(++i==l.end())
        return *l.begin();
    else
        return *i;
}

double Domain::CurveLength(Domain::li i)
{
    double l=0,k=0;
    do
    {
        gPOINT p1=splines[i->misc->spline_index1].at(k);
        k+=0.1;
        gPOINT p2=splines[i->misc->spline_index1].at(k);
        l+=length(p1,p2);
    }while(k<0.99);
    return l;
}

void Triangulation::triangulate(double maxarea)
{

    {
        list<gPOINT> testpoints;
        for(list<gPOINT>::iterator i=domain.points.begin();
            i!=domain.points.end();i++)
        {
            if(i->curved())
                testpoints.push_back(domain.splines[i->misc
                    ->spline_index1].at(0.90));
            testpoints.push_back(*i);
            if(i->misc)
                if(i->misc->spline_index2!=-1)
                    testpoints.push_back(domain.splines[i->
                        misc->spline_index2].at(0.10));
        }
        triangulateio tin,tout;memset(&tin,0,sizeof(tin));
           memset(&tout,0,sizeof(tout));
        //fill out points info
        int tsize=testpoints.size();

        tin.numberofpoints=tsize;
        tin.pointlist=new double[tsize*2];
        list<gPOINT>::iterator ti2=testpoints.begin();
        for(int i1=0;ti2!=testpoints.end();i1++,ti2++)
        {
            tin.pointlist[i1*2]=(*ti2).x;
            tin.pointlist[i1*2+1]=(*ti2).y;
        }
        //fill out segments info
        tin.numberofsegments=tsize;
        tin.segmentlist=new int[tsize*2];
        for(int i=0;i<tsize;i++)
        {
            tin.segmentlist[i*2]=i;
            tin.segmentlist[i*2+1]=i+1;
        }
        tin.segmentlist[2*tsize-1]=0;
        //...create triangulation
        ::triangulate("pz",&tin,&tout,0);
        delete[] tin.pointlist;
        delete[] tin.segmentlist;

        triangles.resize(tout.numberoftriangles);
        int nPoints=tout.numberofpoints;
        int i1=0;
        for(list<CTriangle>::iterator i2=triangles.begin();
            i2!=triangles.end();i1++,i2++)
        {
            int v1=tout.trianglelist[i1*3],v2=tout.
                trianglelist[i1*3+1],v3=tout.trianglelist[i1
                *3+2];
            *i2=CTriangle(v1,v2,v3);
        }
        //fill out points vector
        points.resize(nPoints);
        for(int i=0;i<nPoints;i++)
            points[i]=gPOINT(tout.pointlist[i*2],tout.
                pointlist[i*2+1]);
        CTriangle::v=points;

        free(tout.trianglelist);
        free(tout.pointlist);
        free(tout.pointmarkerlist);
        free(tout.segmentlist);
        free(tout.segmentmarkerlist);
        //tri=true;return;//use this to see how Delaneau
            triangulation looks like
    }
    //how many points originally
    int orign=domain.points.size();
    int* o=new int[orign];//these are to locate the
        position of original points in the new list
```

```
int o_i1=0,o_i2=0;
//subdivide curved segments
    //////////////////////////////
//this part is a bit of a hack: the triangulator doesn'
    t support curved domains, just PSLGs.
//the only problem with this is when segment is refined
    , the new point is misplaced if segment is curved.
//solution: make sure curved segments are not refined
    by refining them beforehand small enough.
//this thing actually works all the time and it should
    check if it doesn't.
//if pre-refined segment is refined, it's not all that
    bad, because it's so small, it's almost a line.
//it would be cool to catch this and just move the
    point.
//the sizes are allright, so it will not form weird
    triangles.
//just move it to splines[i].at(t) where t is computed
    based on t1 and t2-coordinates of endpoints within
    spline
//and the a:b(:c...) ratio in which the segment was
    divided.
double edge=sqrt(maxarea);
list<gPOINT> newpoints;
for(list<gPOINT>::iterator i=domain.points.begin();i!=
    domain.points.end();i++)
     if(i->curved())
     {
          int parts=domain.CurveLength(i)/edge;
          double shift=1.0/parts;
          double s=shift;

          for(int k=0;k<parts-1;k++,s+=shift)
          {
               gPOINT pt=domain.splines[i->misc->
                   spline_index1].at(s);
               pt.misc->spline_index1=i->misc->
                   spline_index1;
               pt.misc->t1=s;
               pt.misc->border=i->border();
               newpoints.push_back(pt);
          }
          //make this more neat
          i->misc->t1=1;
          newpoints.push_back(*i);


          o_i1+=parts-1;
          o[o_i2++]=o_i1++;
```

```
        }
        else
        {
            o[o_i2++]=o_i1++;
            newpoints.push_back(*i);
        }
    ///////////////////////////////

    //maybe check if it's already triangulated
    triangulateio in,out,out1;memset(&in,0,sizeof(in));
        memset(&out,0,sizeof(out));memset(&out1,0,sizeof(
        out1));
    //get the number of points
    int shift1=newpoints.size();

    //this block addresses the following issue:
    //the triangulator doesn't have option to not put two
        edges of same triangle on border.
    //this can be fixed if at each corner point we insert a
         segment of the bisectrissa.
    //TODO:the problem is that i don't know which of two
        angles to bisect(convex/concave issue).
    //that's why this code works just for convex domains.
    //The universal solution would be to triangulate domain
         and check if the new point falls in there.
    //the unrefined triangulation shouldn't take much time
        to compute
    //YES! it works

    list<gPOINT>::iterator i=domain.points.begin();
    for(int j=0;j<orign;i++,j++)
    {
        gPOINT p=(*i);
        gPOINT p1=prev(domain.points,i);
        gPOINT p2=next(domain.points,i);

        gPOINT m1(p.x*0.95+p1.x*0.05,p.y*0.95+p1.y*0.05);
        gPOINT m2(p.x*0.95+p2.x*0.05,p.y*0.95+p2.y*0.05);
        gPOINT n(0.5*(m1.x+m2.x),0.5*(m1.y+m2.y));
        gPOINT n1=p+(n+p*(-1))*(edge/length(p,n));
        gPOINT n2=p+(n*(-1)+p)*(edge/length(p,n));

        list<CTriangle>::iterator i=triangles.begin();
        for(;i!=triangles.end();i++)
            if(i->Holds(n1.x,n1.y))
            {
                newpoints.push_back(n1);
                break;
            }
            else
                if(i->Holds(n2.x,n2.y))
                {
                    newpoints.push_back(n2);
                    break;
                };
        _ASSERTE(i!=triangles.end());
        //newpoints.push_back(n);
    }
    m_ptList=newpoints;


    int size=m_ptList.size();
    if(size<3)
        return;
    //fill out points info
    in.numberofpoints=size;
    in.pointlist=new double[size*2];
    in.pointmarkerlist=new int[size];memset(in.
        pointmarkerlist,0,size*sizeof(int));
    vector<gPOINT> ptvector;ptvector.resize(m_ptList.size()
        );
    list<gPOINT>::iterator i2=m_ptList.begin();
    for(int i1=0;i2!=m_ptList.end();i1++,i2++)
        ptvector[i1]=*i2;
    i2=m_ptList.begin();
    for(int i1=0;i2!=m_ptList.end();i1++,i2++)
    {
        in.pointlist[i1*2]=(*i2).x;
        in.pointlist[i1*2+1]=(*i2).y;
        in.pointmarkerlist[i1]=10000+i1;
    }
    //fill out segments info
    in.numberofsegments=size;
    in.segmentlist=new int[size*2];
    in.segmentmarkerlist=new int[size];//segment markers
        are responsible for boundary type
    i2=m_ptList.begin();
    {
        size-=orign;
        int i=0;
        for(;i<size-1;i++)
        {
            in.segmentlist[i*2]=i;
            in.segmentlist[i*2+1]=i+1;
            in.segmentmarkerlist[i]=(*++i2).border()+777;
        }
        in.segmentlist[i*2]=i;
        in.segmentlist[i*2+1]=0;
        in.segmentmarkerlist[i]=m_ptList.begin()->border()
            +777;

        for(int i=0;i<orign;i++)
        {
            in.segmentlist[size*2+2*i]=o[i];
            in.segmentlist[size*2+2*i+1]=size+i;
            in.segmentmarkerlist[size+i]=101;
        }

    }
    delete[]o;

        char szBuf[100];
        sprintf(szBuf,"pa%fz",maxarea);
        //...create triangulation
        ::triangulate(szBuf,&in,&out1,0);
        delete[] in.pointlist;
        delete[] in.segmentlist;
        delete[] in.segmentmarkerlist;
        //...refine triangulation
        ::triangulate("pqze",&out1,&out,0);
        free(out1.trianglelist);
        free(out1.pointlist);
        free(out1.pointmarkerlist);
        free(out1.segmentlist);
        free(out1.segmentmarkerlist);

        //fill out triangle indices
        triangles.resize(out.numberoftriangles);
        int t1=0,t2=0,t3=0,t4=0,t5=0,t6=0;
        int nPoints=out.numberofpoints;
        int i1=0;
        for(list<CTriangle>::iterator i2=triangles.begin();
            i2!=triangles.end();i1++,i2++)
        {
            int v1=out.trianglelist[i1*3],v2=out.
                trianglelist[i1*3+1],v3=out.trianglelist[i1
                *3+2];
            int m1,m2,m3;

            for(int i=0;i<out.numberofedges;i++)
            {
                if(out.edgelist[2*i]==v1)
                {
                    if(out.edgelist[2*i+1]==v2)
                        m3=i,t1++;
                    else
                        if(out.edgelist[2*i+1]==v3)
                        m2=i , t2++;
                }
                else
                    if(out.edgelist[2*i]==v2)
                    {
                        if(out.edgelist[2*i+1]==v1)
                            m3=i , t3++;
                        else
                            if(out.edgelist[2*i+1]==v3)
                                m1=i , t4++;
                    }
                    else
                        if(out.edgelist[2*i]==v3)
                        {
                            if(out.edgelist[2*i+1]==v1)
                                m2=i , t5++;
                            else
                                if(out.edgelist[2*i+1]==v2)
                                    m1=i , t6++;
                        }
            }
            *i2=CTriangle(v1,v2,v3,m1+nPoints,m2+nPoints,m3
                +nPoints);
        }
        //_ASSERTE(out.numberofsegments==m_ptList.size());
        int counter=0;
        if(out.numberofsegments!=m_ptList.size())//
            triangulator has refined a curved segment
        {
            for(int i=0;i<out.numberofsegments;i++)
            {
                if(out.pointmarkerlist[out.segmentlist[2*i
                    ]]<1000)
                    counter++;
                if(out.pointmarkerlist[out.segmentlist[2*i
                    +1]]<1000)
                    counter++;
            }
            //MessageBox(0,0,0,0);
        }
        //_ASSERTE(out.numberofsegments==counter/2+m_ptList
            .size());
        //fill out points vector
        points.resize(nPoints);
        for(int i=0;i<nPoints;i++)
            if(out.pointmarkerlist[i]>=10000)
                points[i]=ptvector[out.pointmarkerlist[i
                    ]-10000];
            else
```

```
            points[i]=gPOINT(out.pointlist[i*2],out.
                pointlist[i*2+1],out.pointmarkerlist[i
                ]-777);
    segments.clear();
    //cycle through all edges and segment
    //when match is found, CSegment is created with 3
        degrees of freedom:
    //two correspond to number of vertices
    //one correspond to number of edge that coincides
        with segment
    for(int i=0;i<out.numberofsegments;i++)
        if(out.segmentmarkerlist[i]==101)
            continue;
        else
        for(int j=0;j<out.numberofedges;j++)
            if(out.edgelist[2*j]==out.segmentlist[2*i])
            {
                if(out.edgelist[2*j+1]==out.segmentlist
                    [2*i+1])
                {
                    segments.push_back(CSegment(out.
                        segmentlist[i*2],out.segmentlist
                        [i*2+1],j+nPoints,out.
                        segmentmarkerlist[i]-777));
                    break;
                }
            }
            else
                if(out.edgelist[2*j]==out.segmentlist
                    [2*i+1])
                    if(out.edgelist[2*j+1]==out.
                        segmentlist[2*i])
                    {
                        segments.push_back(CSegment(out
                            .segmentlist[i*2],out.
                            segmentlist[i*2+1],j+nPoints
                            ,out.segmentmarkerlist[i
                            ]-777));
                        break;
                    };
    tri=true;
    free(out.trianglelist);
    free(out.pointlist);
    free(out.pointmarkerlist);
    free(out.segmentlist);
    free(out.segmentmarkerlist);
    free(out.edgelist);
    free(out.edgemarkerlist);
    delete[] in.pointmarkerlist;
```

```
}

//////CTriangle//////////////////////////////
CTriangle::approximation_type CTriangle::type=CTriangle::
    quadratic;
CTriangle::quadrature_type CTriangle::qtype=CTriangle::
    gauss2;

fptrv CTriangle::P1_gradN[3]={CTriangle::P1_gradN1,
    CTriangle::P1_gradN2,CTriangle::P1_gradN3};
fptr  CTriangle::P1_N[3]={CTriangle::P1_N1,CTriangle::P1_N2
    ,CTriangle::P1_N3};
fptrv CTriangle::P2_gradN[6]={CTriangle::P2_gradN1,
    CTriangle::P2_gradN2,CTriangle::P2_gradN3,
                                    CTriangle::P2_gradN4,
                                        CTriangle::P2_gradN5,
                                        CTriangle::P2_gradN6};
fptr CTriangle::P2_N[6]={CTriangle::P2_N1,CTriangle::P2_N2,
    CTriangle::P2_N3,
                                    CTriangle::P2_N4,CTriangle::
                                        P2_N5,CTriangle::P2_N6};

gPOINT CTriangle::p4(gPOINT p1, gPOINT p2)
{
     if(p1.misc->spline_index1==p2.misc->spline_index1)
          return splines[p1.misc->spline_index1].at(0.5*(p1.
              misc->t1+p2.misc->t1));
     else
     if(p1.misc->spline_index1==p2.misc->spline_index2)
          return splines[p1.misc->spline_index1].at(0.5*(p1.
              misc->t1+p2.misc->t2));
     else
     if(p1.misc->spline_index2==p2.misc->spline_index1)
          return splines[p1.misc->spline_index2].at(0.5*(p1.
              misc->t2+p2.misc->t1));
     else
          throw int();
     return gPOINT(0,0);
}

void CTriangle::Contribute(double** K,double* Q)
{
     double x1=v[i1].x,x2=v[i2].x,x3=v[i3].x;
     double y1=v[i1].y,y2=v[i2].y,y3=v[i3].y;
     double Bk[2][2]={{x2-x1,x3-x1},
                      {y2-y1,y3-y1}};
     dx=dy=0;
     if(type==isoparametric)
     {
        //first a little check:
        //make sure only not all three verts are curved
        _ASSERTE((v[i1].curved() && v[i2].curved() && v[i3
            ].curved())==false);

        gPOINT p;
        if(v[i1].curved() && v[i2].curved())
        {
            p=p4(v[i1],v[i2]);
            dx=p.x-0.5*(v[i1].x+v[i2].x);
            dy=p.y-0.5*(v[i1].y+v[i2].y);
        }
        if(v[i1].curved() && v[i3].curved())
        {
            p=p4(v[i1],v[i3]);
            dx=p.x-0.5*(v[i1].x+v[i3].x);
            dy=p.y-0.5*(v[i1].y+v[i3].y);
        }
        if(v[i2].curved() && v[i3].curved())
        {
            p=p4(v[i2],v[i3]);
            dx=p.x-0.5*(v[i2].x+v[i3].x);
            dy=p.y-0.5*(v[i2].y+v[i3].y);
        }
    }

    //   use this Gauss-Legendre quadrature
    double qx2[]={0.211324865,0.211324865,
        0.788675134,0.788675134};double* x=qx2;
    double qy2[]={0.166666667,0.622008467,
        0.044658198,0.166666667};double* y=qy2;
    double qc2[]={0.197168783,0.197168783,
        0.052831216,0.052831216};double* c=qc2;

    double qx3[]={0.112701665,0.112701665,0.112701665,
        0.500000000,0.500000000,0.500000000,
        0.887298334,0.887298334,0.887298334};
    double qy3[]={0.100000000,0.443649167,0.787298334,
        0.056350832,0.250000000,0.443649167,
        0.012701665,0.056350832,0.100000000};
    double qc3[]={0.068464377,0.109543004,0.068464377,
        0.061728395,0.098765432,0.061728395,
        0.008696116,0.013913785,0.008696116};

    int q_n=4;

    if(qtype==gauss3)
    {
        q_n=9;
```

```
            x=qx3;y=qy3;c=qc3;
        }

        //adjust things according to approximation type
        int l_ind[3]={i1,i2,i3};
        int q_ind[6]={i1,i2,i3,m1,m2,m3};
        int* ind=q_ind;
        int n=6;
        fptr* N=P2_N;
        fptrv* gradN=P2_gradN;
        if(type==linear)
        {
            ind=l_ind;
            n=3;
            N=P1_N;
            gradN=P1_gradN;
        }

        matr Bk_T(Bk[1][1]/detBk(),-Bk[1][0]/detBk(),-Bk[0][1]/
            detBk(), Bk[0][0]/detBk());
        using namespace HeatConductionProblem;
        for(int m=0;m<q_n;m++)
        {
            //quadrature points on real triangle:
            double xm=xFk(x[m],y[m]),ym=yFk(x[m],y[m]);
            matr k(k11(xm,ym),k12(xm,ym),
                   k12(xm,ym),k22(xm,ym));
            double detB=abs(detBk());
            if(type==isoparametric)
            {
                matr Bk=BK(xm,ym);
                detB=abs(det(Bk));
                Bk_T=invt(Bk);
            }
            for(int i=0;i<n;i++)
            {
                for(int j=0;j<n;j++)
                    K[ind[i]][ind[j]]+=c[m] * detB *( (Bk_T*
                        gradN[i](x[m],y[m])) * (k*(Bk_T*gradN[j
                        ](x[m],y[m]))) );
                Q[ind[i]]+=c[m] * detB * f(xm,ym)*N[i](x[m],y[m
                    ]);
            }
        }
}

void CTriangle::ContributeMass(double** M)
{
        double x1=v[i1].x,x2=v[i2].x,x3=v[i3].x;
double y1=v[i1].y,y2=v[i2].y,y3=v[i3].y;
double Bk[2][2]={{x2-x1,x3-x1},
                 {y2-y1,y3-y1}};
dx=dy=0;
if(type==isoparametric)
{
    //first a little check:
    //make sure only not all three verts are curved
    _ASSERTE((v[i1].curved() && v[i2].curved() && v[i3
        ].curved())==false);

    gPOINT p;
    if(v[i1].curved() && v[i2].curved())
    {
        p=p4(v[i1],v[i2]);
        dx=p.x-0.5*(v[i1].x+v[i2].x);
        dy=p.y-0.5*(v[i1].y+v[i2].y);
    }
    if(v[i1].curved() && v[i3].curved())
    {
        p=p4(v[i1],v[i3]);
        dx=p.x-0.5*(v[i1].x+v[i3].x);
        dy=p.y-0.5*(v[i1].y+v[i3].y);
    }
    if(v[i2].curved() && v[i3].curved())
    {
        p=p4(v[i2],v[i3]);
        dx=p.x-0.5*(v[i2].x+v[i3].x);
        dy=p.y-0.5*(v[i2].y+v[i3].y);
    }
}

//   use this Gauss-Legendre quadrature
double x[]={0.211324865,0.211324865,
    0.788675134,0.788675134};
double y[]={0.166666667,0.622008467,
    0.044658198,0.166666667};
double c[]={0.197168783,0.197168783,
    0.052831216,0.052831216};
const int q_n=4;

//adjust things according to approximation type
int l_ind[3]={i1,i2,i3};
int q_ind[6]={i1,i2,i3,m1,m2,m3};
int* ind=q_ind;
int n=6;
fptr* N=P2_N;
fptrv* gradN=P2_gradN;
if(type==linear)
```

```
    {
        ind=l_ind;
        n=3;
        N=P1_N;
        gradN=P1_gradN;
    }

    matr Bk_T(Bk[1][1]/detBk(),-Bk[1][0]/detBk(),-Bk[0][1]/
        detBk(), Bk[0][0]/detBk());
    using namespace HeatConductionProblem;
    for(int m=0;m<q_n;m++)
    {
        //quadrature points on real triangle:
        double xm=xFk(x[m],y[m]),ym=yFk(x[m],y[m]);
        matr k(k11(xm,ym),k12(xm,ym),
               k12(xm,ym),k22(xm,ym));
        double detB=abs(detBk());
        if(type==isoparametric)
        {
            matr Bk=BK(xm,ym);
            detB=abs(det(Bk));
        }
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                M[ind[i]][ind[j]]+=c[m] * detB * N[i](x[m],
                    y[m]) * N[j](x[m],y[m]);
    }
}

//////Domain///////////////////////////////

void Domain::Draw(CDC* pDC,CTriangleView* view)
{
    static COLORREF clrLine=RGB(255,0,255);
    static COLORREF clrActiveLine=RGB(0,255,125);
    static COLORREF clrVertex=RGB(0,0,0);
    static COLORREF clrActiveVertex=RGB(255,0,0);
    static COLORREF clrMidpoint=RGB(0,40,255);
    static CBrush brVertex(RGB(0,0,0));
    static CBrush brActiveVertex(RGB(255,0,0));
    static CBrush brMidpoint(RGB(0,40,255));
    if(points.size()==0)
        return;
    //repaint everything
    CPen pen;pen.CreatePen(PS_SOLID,3,clrLine);CPen*pOldPen
        =pDC->SelectObject(&pen);

    li i=points.begin();
    pDC->MoveTo(view->RT(*i));
```

```
for(;i!=points.end();i++)
{
    if(i->misc)
    {
        int t=i->misc->spline_index1;
        if(t!=-1)
        {
            pDC->MoveTo(view->RT(splines[t].at(0)));
            for(double k=0.1;k<=1.01;k+=0.1)
                pDC->LineTo(view->RT(splines[t].at(k)))
                    ;
            continue;
        }
    }
    pDC->LineTo(view->RT(*i));
}
if(points.begin()->misc==0 || points.begin()->misc->
    spline_index1==-1)
    pDC->LineTo(view->RT(*points.begin()));

CPen penDash;penDash.CreatePen(PS_SOLID,3,clrActiveLine
    );pDC->SelectObject(&penDash);
pDC->MoveTo(view->RT(*currVertex));

li to=currVertex;
if(currVertex==points.begin())
    to=points.end();
--to;
if(currVertex->misc)
{
    int t=currVertex->misc->spline_index1;
    if(t!=-1)
    {
        pDC->MoveTo(view->RT(splines[t].at(0)));
        for(double k=0.1;k<=1.01;k+=0.1)
            pDC->LineTo(view->RT(splines[t].at(k)));
    }
    else
        pDC->LineTo(view->RT(*to));
}
else
    pDC->LineTo(view->RT(*to));

for(li i=points.begin();i!=points.end();i++)
    pDC->FillRect(CRect(view->RT(*i)-CSize(4,4),CSize
        (8,8)),&brVertex);
pDC->FillRect(CRect(view->RT(*currVertex)-CSize(4,4),
    CSize(8,8)),&brActiveVertex);
if(currVertex->misc)
```

```
            if(currVertex->misc->spline_index1!=-1)
            {
                gPOINT m=splines[currVertex->misc->
                    spline_index1].m;
                pDC->FillRect(CRect(view->RT(m)-CSize(4,4),
                    CSize(8,8)),&brMidpoint);
            }
        pDC->SelectObject(pOldPen);
}

void Domain::SelectVertexAt(gPOINT at)
{
        li i=points.begin();currVertex=i;
        double minl=length(*i,at);
        for(;i!=points.end();i++)
            if(length(at,*i)<minl)
            {
                minl=length(at,*i);
                currVertex=i;
            };
}

void Domain::SelectCurveAt(gPOINT at)
{
        li i1=points.begin();
        li i2=i1;i2++;
        currVertex=i2;
        double minl=length(*i1,at)+length(*i2,at)-length(*i1,*
            i2);
        while(++i2!=points.end())
        {
            i1++;
            double l=length(*i1,at)+length(*i2,at)-length(*i1,*
                i2);
            if(l<minl)
            {
                minl=l;
                currVertex=i2;
            }
        }
        i1=points.begin();
        i2=--points.end();
        double l=length(*i1,at)+length(*i2,at)-length(*i1,*i2);
        if(l<minl)
            currVertex=i1;
}

void Domain::AddVertex(gPOINT v)
{
        if(points.empty())
        {
            points.push_back(v);
            currVertex=points.begin();
        }
        else
        {
            currVertex=points.insert(currVertex,v);
            if(currVertex->misc)
                if(currVertex->misc->spline_index1!=-1)
                    splines[currVertex->misc->spline_index1].t=
                        v;
        }
}
void Domain::DeleteVertex()
{
        int n=points.size();
        if(n==0)
            return;
        if(n==1)
        {
            points.clear();
            return;
        }
        if(currVertex==--(points.end()))
        {
            li t=currVertex;
            --currVertex;
            points.erase(t);
            return;
        }
        currVertex=points.erase(currVertex);

}

void Domain::NextVertex()
{
        if(++currVertex==points.end())
            currVertex=points.begin();
}

void Domain::PrevVertex()
{
        if(currVertex==points.begin())
            currVertex=--points.end();
        else
            --currVertex;
}
```

```
void Domain::clear()
{
    points.clear();
    splines.clear();
}

void Domain::MoveVertex(gPOINT v)
{
    currVertex->x=v.x;
    currVertex->y=v.y;
    if(currVertex->misc)
    {
        if(currVertex->misc->spline_index1!=-1)
            splines[currVertex->misc->spline_index1].s=v;
        if(currVertex->misc->spline_index2!=-1)
            splines[currVertex->misc->spline_index2].t=v;
    }
}

void Domain::AddSpline(gPOINT midpoint, li* at)
{
    li p1=at? *at:currVertex;
    li p2=p1;if(p2==points.begin())p2=points.end();p2--;

    if(p1->misc==0)
        p1->misc=new gPOINT::aux(0,splines.size());
    else
        p1->misc->spline_index1=splines.size();
    if(p2->misc==0)
        p2->misc=new gPOINT::aux(0,-1,0,splines.size());
    else
        p2->misc->spline_index2=splines.size();


    splines.push_back(Spline(*p1,midpoint,*p2));
}

void Domain::SetMidpoint(gPOINT at)
{
    if(currVertex->curved())
        splines[currVertex->misc->spline_index1].m=at;
    else
        AddSpline(at);
}

void Domain::ToggleCurved()
{
    if(currVertex->misc)
        if(currVertex->misc->spline_index1!=-1)
        {
            //splines.erase
            currVertex->misc->spline_index1=-1;
            return;
        }
    SetMidpoint(*currVertex);
}

bool Domain::GetCurved()
{
    if(currVertex->misc==0)
        return false;
    if(currVertex->misc->spline_index1!=-1)
        return true;
    return false;
}
```

```
#pragma once
#include "afxwin.h"


// CSubdomainDialog dialog

class CSubdomainDialog : public CDialog
{
    DECLARE_DYNAMIC(CSubdomainDialog)

public:
    CSubdomainDialog(CWnd* pParent = NULL);   // standard
        constructor
    virtual ~CSubdomainDialog();

// Dialog Data
    enum { IDD = IDD_SUBDOMAINDIALOG };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    //
        DDX/DDV support

    DECLARE_MESSAGE_MAP()
public:
    static double mu;
    static CString k11;
    static CString k22;
    static CString k12;
    static CString f;
    afx_msg void OnBnClickedOk();
    static void update();
};
```

```
// SubdomainDialog.cpp : implementation file
//

#include "stdafx.h"
#include "MFCtriangle.h"
#include "SubdomainDialog.h"
#include "stage.h"


// CSubdomainDialog dialog

IMPLEMENT_DYNAMIC(CSubdomainDialog, CDialog)

using namespace HeatConductionProblem;
char* conv(const wchar_t* s );

double  CSubdomainDialog::mu;
CString CSubdomainDialog::k11;
CString CSubdomainDialog::k22;
CString CSubdomainDialog::k12;
CString CSubdomainDialog::f;

CSubdomainDialog::CSubdomainDialog(CWnd* pParent /*=NULL*/)
    : CDialog(CSubdomainDialog::IDD, pParent)
{
}

CSubdomainDialog::~CSubdomainDialog()
{
}

void CSubdomainDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Text(pDX,IDC_edit_mu,mu);
    DDX_Text(pDX,IDC_edit_k11,k11);
    DDX_Text(pDX,IDC_edit_k22,k22);
    DDX_Text(pDX,IDC_edit_k12,k12);
    DDX_Text(pDX,IDC_edit_Q,f);
}


BEGIN_MESSAGE_MAP(CSubdomainDialog, CDialog)
    ON_BN_CLICKED(IDOK, &CSubdomainDialog::OnBnClickedOk)
END_MESSAGE_MAP()


// CSubdomainDialog message handlers
```

```
void CSubdomainDialog::update()
{
    int e=K11.Parse(string(conv(k11)),"x,y");K11.Optimize()
        ;
    e=K12.Parse(string(conv(k12)),"x,y");     K12.Optimize()
        ;
    e=K22.Parse(string(conv(k22)),"x,y");     K22.Optimize()
        ;
    e=F.Parse(string(conv(f)),"x,y");         F.Optimize();
}

void CSubdomainDialog::OnBnClickedOk()
{
    OnOK();
    update();
}
```

```
// TriangleDoc.h : interface of the CTriangleDoc class
//
#include "stage.h"
using namespace std;
#pragma once
//problem:              dT/dt=  (kT)+f
class CTriangleDoc : public CDocument
{
protected: // create from serialization only
    CTriangleDoc();
    DECLARE_DYNCREATE(CTriangleDoc)

// Attributes
public:
    //list<gPOINT> m_ptList;            //list of linked
        points
    Domain domain;
    vector<gPOINT> m_ptVector;       //vector of points of
        triangulation, valid only when m_bTri
    list<CTriangle> m_triList;       //list of linked
        triangle indexes
    list<CSegment>m_segList;         //list of segments

    problem p;

    vector<gBoundary*> m_bndVector; //vector of boundaries
    double m_dt;
    bool m_bTri;                     //true if has been
        triangulated
    bool m_bSolved;                  //true if solution is
        valid
    double *u;                       //solution in each
        point
    vector<double*> U;//solution in each point at each time
    //preferences
    double m_Left;
    double m_Top;
    double m_Right;
    double m_Bottom;
    double m_Area;
    bool m_bBoundaryLabels;
// Operations
public:
    gPOINT Solve();//solves, returns min and max result
    typedef list<gPOINT>::iterator li;
    double Eval(double x,double y);
    double Eval(double x,double y,double t);
// Overrides
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Implementation
public:
    virtual ~CTriangleDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    void ClearBoundaryInfo();
// Generated message map functions
protected:
    DECLARE_MESSAGE_MAP()
};
```

```
// TriangleDoc.cpp : implementation of the CTriangleDoc
    class
//

#include "stdafx.h"
#include "MFCtriangle.h"
#include "TriangleDoc.h"
#include "rigid.h"
#include "SubdomainDialog.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#endif
//TODO
//HeatConductionProblem CTriangle::problem(0,0,0,0)
    ;//(1,3,5,1);
vector<gPOINT> dummy1;
vector<gPOINT>& CTriangle::v=dummy1;
vector<Spline> dummy2;
vector<Spline>& CTriangle::splines=dummy2;
double** gBoundary::K;
double*  gBoundary::Q;
vector<gPOINT>& gBoundary::points=vector<gPOINT>();
// CTriangleDoc

IMPLEMENT_DYNCREATE(CTriangleDoc, CDocument)

BEGIN_MESSAGE_MAP(CTriangleDoc, CDocument)
END_MESSAGE_MAP()


// CTriangleDoc construction/destruction

CTriangleDoc::CTriangleDoc()
: m_bTri(false),m_Area(0.005),m_Left(0),m_Bottom(0),m_Right
    (2),m_Top(2),m_bBoundaryLabels(false)
{
}

void CTriangleDoc::ClearBoundaryInfo()
{
    vector<gBoundary*>::iterator start=m_bndVector.begin(),
        end=m_bndVector.end();
    for(;start!=end;start++)
        delete (*start);
    m_bndVector.clear();
}
```

```
CTriangleDoc::~CTriangleDoc()
{
    ClearBoundaryInfo();
}

using namespace HeatConductionProblem;
char* conv(const wchar_t* s );

BOOL CTriangleDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;
    m_bTri=false;
    m_bSolved=false;
    domain.clear();
    m_triList.clear();
    m_segList.clear();
    //...delete boundary info
    ClearBoundaryInfo();
    //...fill boudnary info with defaults
    m_bndVector.push_back(new TemperatureBoundary(0));
    m_bndVector.push_back(new HeatFluxBoundary(0,0,0));
    m_bndVector.push_back(new HeatFluxBoundary(0,1,0));
    //...fill subdomain info with defaults
    CSubdomainDialog::k11="1";
    CSubdomainDialog::k12="0";
    CSubdomainDialog::k22="1";
    CSubdomainDialog::f="2*pi*pi*sin(pi*x)*sin(pi*y)";
    double pi=3.14159265358979323846264338327950;
    F.AddConstant("pi",pi);
    CSubdomainDialog::mu=1;
    K11.Parse(string(conv(CSubdomainDialog::k11)),"x,y");
        K11.Optimize();
    K12.Parse(string(conv(CSubdomainDialog::k12)),"x,y");
        K12.Optimize();
    K22.Parse(string(conv(CSubdomainDialog::k22)),"x,y");
        K22.Optimize();
    F.Parse(string(   conv(CSubdomainDialog::f))   ,"x,y");F.
        Optimize();
    return TRUE;
}

// CTriangleDoc serialization

void CTriangleDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        //...store view preferences,maximum triangle area
        ar<<m_Left<<m_Top<<m_Right<<m_Bottom<<m_Area;
        ar<<m_bTri;
        //...store domain
        ar<<domain.points.size();
        for(list<gPOINT>::iterator i=domain.points.begin();
            i!=domain.points.end();i++)
        {
            ar<<(*i).x<<(*i).y<<(*i).border();
            bool c=(*i).curved();ar<<c;
            if(c)
                ar<<(*i).misc->spline_index1<<(*i).misc->t1
                    <<(*i).misc->spline_index2<<(*i).misc->
                    t2;
            else
                ar<<int(-1)<<double(0)<<int(-1)<<double(0);
        }
        ar<<int(CTriangle::type);
        ar<<domain.splines.size();
        for(int i=0;i<domain.splines.size();i++)
            ar<<domain.splines[i].s.x<<domain.splines[i].s.
                y
              <<domain.splines[i].m.x<<domain.splines[i].m.
                  y
              <<domain.splines[i].t.x<<domain.splines[i].t.
                  y;
        //...store subdomain info
        ar<<CSubdomainDialog::k11<<CSubdomainDialog::k12<<
            CSubdomainDialog::k22<<CSubdomainDialog::f;
        //...store boundary info
        ar<<m_bndVector.size();
        vector<gBoundary*>::iterator i=m_bndVector.begin(),
            end=m_bndVector.end();
        for(;i!=end;++i)
            (*i)->Serialize(ar);
    }
    else
    {
        //...read view preferences,maximum triangle area
        ar>>m_Left>>m_Top>>m_Right>>m_Bottom>>m_Area;
        ar>>m_bTri;
        //...read domain
        int size;
        ar>>size;
        domain.clear();
        m_triList.clear();
        m_segList.clear();
        ClearBoundaryInfo();
        for(int i=0;i<size;i++)
        {
            double x,y;
            int border;
            bool curved;
            int spline_index1,spline_index2;
            double t1,t2;
            ar>>x>>y>>border>>curved>>
                spline_index1>>t1>>spline_index2>>t2;;

            domain.points.push_back(gPOINT(x,y,border,
               spline_index1,t1,spline_index2,t2));
        }
        domain.currVertex=domain.points.begin();
        int tp;ar>>tp;CTriangle::type=CTriangle::
           approximation_type(tp);
        ar>>size;domain.splines.resize(size);
        for(int i=0;i<size;i++)
        {
            double sx,sy,mx,my,tx,ty;
            ar>>sx>>sy>>mx>>my>>tx>>ty;
            domain.splines[i]=Spline(gPOINT(sx,sy),gPOINT(
               mx,my),gPOINT(tx,ty));
        }
        //...read subdomain info
        ar>>CSubdomainDialog::k11>>CSubdomainDialog::k12>>
           CSubdomainDialog::k22>>CSubdomainDialog::f;
        CSubdomainDialog::update();
        //...read boundary info
        ar>>size;
        for(int i=0;i<size;++i)
        {
            int kind;ar>>kind;
            gBoundary* boundary;
            switch(kind)
            {
            case 0://temperature
                boundary=new TemperatureBoundary(0);
                break;
            case 1://flux
                boundary=new HeatFluxBoundary(0,0,0);
                break;
            default:
                _ASSERTE(1);
            }
            boundary->Serialize(ar);
            m_bndVector.push_back(boundary);
        }
    }
}
```

```
// CTriangleDoc diagnostics

#ifdef _DEBUG
void CTriangleDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CTriangleDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG
// CTriangleDoc commands
gPOINT CTriangleDoc::Solve()
{
    //allocate memory
    int n=m_ptVector.size();
    if(n<3)
    {
        MessageBox(0,_T("No␣geometry␣defined!"),_T("error!"
            ),0);
        return gPOINT(0,0);
    }
    //for p2, number of d.o.f. is (number of points)+(
       number of edges)
    //number of edges=(3*(number of triangles)+number of
       boundary edges)/2

    //for heat with p2 or isoparametric p2
    if(CTriangle::type!=CTriangle::linear)
        n=n+(3*m_triList.size()+m_segList.size())/2;

    double** K=new double*[n];
    for(int i=0;i<n;i++)
    {
        K[i]=new double[n];
        memset(K[i],0,n*sizeof(double));
    }
    double* Q=new double[n];
    memset(Q,0,n*sizeof(double));
    //let CTriangle know location of points
    CTriangle::v=m_ptVector;
    CTriangle::splines=domain.splines;
    //assemble matrices
    for(list<CTriangle>::iterator i=m_triList.begin();i!=
        m_triList.end();i++)
        (*i).Contribute(K,Q);
    //accommodate for boundaries
    gBoundary::Init(K,Q,m_ptVector);
    for(list<CSegment>::iterator i=m_segList.begin();i!=
        m_segList.end();i++)
         m_bndVector[i->boundary]->Accomodate(i);//this
             function is virtual: either Dirichlet or Newton
    //solve linear system
    gRigid::Graph g(n,K);
    u=g.Solve(Q);
    m_bSolved=true;
    //free memory
    delete[]Q;
    for(int i=0;i<n;i++)
         delete[] K[i];
    delete[]K;
    //...return minimum and maximum values
    n=m_ptVector.size();
    gPOINT res(u[0],u[0]);
    for(int i=1;i<n;i++)
         if(u[i]>res.x)
              res.x=u[i];
         else
              if(u[i]<res.y)
                   res.y=u[i];

    /*res.x=0;
    for(list<CTriangle>::iterator i=m_triList.begin();i!=
        m_triList.end();i++)
         res.x+=i->integral(u);*/

    return res;
}

double CTriangleDoc::Eval( double x,double y )
{
    //task: do the evaluatoin using Besier splines,
    //actually nothing to do here, just leave it up to
        CTriangle
    _ASSERTE(m_bSolved!=false);
    list<CTriangle>::iterator i=m_triList.begin();
    for(;i!=m_triList.end();i++)
         if((*i).Holds(x,y))
              break;
    if(i==m_triList.end())
         return 10e20;
    return (*i).Eval(u,x,y);
}

double CTriangleDoc::Eval(double x,double y,double t)
```

```
{
    double tmax=(U.size()-1)*m_dt;
    if( (t<0)||(t>=tmax) )
    {
        MessageBox(0,_T("t out of range!"),_T("error"),0);
        return 10e20;
    }
    _ASSERTE(m_bSolved==true);
    list<CTriangle>::iterator i=m_triList.begin();
    for(;i!=m_triList.end();i++)
        if((*i).Holds(x,y))
            break;
    if(i==m_triList.end())
        return 10e20;
    double u1=u[(*i).i1];
    double u2=u[(*i).i2];
    double u3=u[(*i).i3];
    int layer=int(t/m_dt);
    double a=t-layer*m_dt;
    u[(*i).i1]=U[layer][(*i).i1]*(1-a)+U[layer+1][(*i).i1]*
        a;
    u[(*i).i2]=U[layer][(*i).i2]*(1-a)+U[layer+1][(*i).i2]*
        a;
    u[(*i).i3]=U[layer][(*i).i3]*(1-a)+U[layer+1][(*i).i3]*
        a;
    double result=(*i).Eval(u,x,y);
    u[(*i).i1]=u1;
    u[(*i).i2]=u2;
    u[(*i).i3]=u3;
    return result;
}
```

```
// TriangleView.h : interface of the CTriangleView class
//

#pragma once
enum ViewMode{VertexInsert=0,PolyInsert,PolyDrag,VertexDrag
    };
#define VIEW_MODE_VERTEX_INSERT 0//insert vertex to split
    the current edge on left mouse button click
#define VIEW_MODE_POLY_INSERT   1//insert poly has been
    pressed - start poly drag operation on left mouse button
     click
#define VIEW_MODE_POLY_DRAG     2//redraw poly on mouse
    move
#define VIEW_MODE_VERTEX_DRAG   3//redraw vertex on mouse
    move
#define VIEW_MODE_MIDPOINT_DRAG 4//redraw spline on mouse
    move
class CTriangleDoc;

class CTriangleView : public CView
{
protected: // create from serialization only
    CTriangleView();
    DECLARE_DYNCREATE(CTriangleView)

// Attributes
public:
    CTriangleDoc* GetDocument() const;
// Operations
public:
    void DrawScale(CDC*pDC,CPoint point,CSize size);
    BOOL UpdateCurrentVertex(gPOINT point);
    gPOINT Transform(CPoint point)
    {
        CRect rect;GetClientRect(&rect);
        int w=rect.Width();
        int h=rect.Height();
        if(!m_bAnisotropic)
        {
            if(w<h)
                h=w;
            else
                w=h;
        }
        return gPOINT(m_Left+point.x*(m_Right-m_Left)/w,
            m_Top+point.y*(m_Bottom-m_Top)/h);
    }
    CPoint RT(gPOINT point)
    {
        CRect rect;GetClientRect(&rect);
        int w=rect.Width();
        int h=rect.Height();
        if(!m_bAnisotropic)
        {
            if(w<h)
                h=w;
            else
                w=h;
        }
        return CPoint((point.x-m_Left)/(m_Right-m_Left)*w,
            /*rect.Height()+*/(point.y-m_Top)/(m_Bottom-
            m_Top)*h);
    }
// Overrides
public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw
        this view
    void Spline(CDC* pDC,gPOINT p1,gPOINT p2);
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnInitialUpdate();
// Implementation
public:
    virtual ~CTriangleView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    CRect rcBounds;
    int nMode;      //VIEW_MODE_...
    //poly insertion attributes
    int nPolySides;
    CPoint ptCenter;
    //appearance attributes
    COLORREF clrVertex;
    COLORREF clrActiveVertex;
    COLORREF clrMidpoint;
    CBrush brVertex;
    CBrush brActiveVertex;
    CBrush brMidpoint;
    COLORREF clrLine;
    COLORREF clrActiveLine;
    //current vertex attributes
    typedef list<gPOINT>::iterator li;
    typedef list<gPOINT>::const_iterator cli;
    //li currVertex,split1,split2;
    //gPOINT gptNewVertex;
    CImage iImage;
    CBitmap bmpBuffer;
public:
    gPOINT minmax;
    CFont fontMinmax;
    //contex menu cache
    static const int menuCurvedid=24238;
    static const int menuBoundaryid=24239;
// Generated message map functions
public:
    afx_msg void OnFileTrianglulate();
    afx_msg void OnFileSolve();
    afx_msg void OnInsertPolygon();
    afx_msg void OnUpdateInsertPolygon(CCmdUI *pCmdUI);
    afx_msg void OnOptionsPreferences();
protected:
    //...mouse
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg BOOL OnMouseWheel(UINT nFlags, short zDelta,
        CPoint pt);
    //...context menu
    afx_msg void OnContextMenu(CWnd*,CPoint);
    afx_msg void OnBoundaryType(UINT nID);
    afx_msg void OnUpdateBoundaryType(CCmdUI* pCmdUI);
    //...misc
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT
        nFlags);
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    DECLARE_MESSAGE_MAP()
protected:
    double m_Left;
    double m_Top;
    double m_Right;
    double m_Bottom;
    bool m_bAnisotropic;
public:
    afx_msg void OnOptionsTimestep();
protected:
    virtual void OnUpdate(CView* /*pSender*/, LPARAM /*
        lHint*/, CObject* /*pHint*/);
};
#ifndef _DEBUG  // debug version in TriangleView.cpp
inline CTriangleDoc* CTriangleView::GetDocument() const
   { return reinterpret_cast<CTriangleDoc*>(m_pDocument); }
#endif
```

```
// TriangleView.cpp : implementation of the CTriangleView
   class
//

#include "stdafx.h"
#include "MFCtriangle.h"

#include "TriangleDoc.h"
#include "TriangleView.h"
#include "triangle.h"
#include "MainFrm.h"
#include "PrefDialog.h"
#include "MatrixEdit.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
#include <vector>


// CTriangleView

IMPLEMENT_DYNCREATE(CTriangleView, CView)

BEGIN_MESSAGE_MAP(CTriangleView, CView)
    ON_WM_LBUTTONDOWN()
    ON_COMMAND(ID_FILE_TRIANGLULATE, &CTriangleView::
        OnFileTrianglulate)
    ON_COMMAND(ID_INSERT_POLYGON, &CTriangleView::
        OnInsertPolygon)
    ON_WM_MOUSEMOVE()
    ON_WM_LBUTTONUP()
    ON_UPDATE_COMMAND_UI(ID_INSERT_POLYGON, &CTriangleView
        ::OnUpdateInsertPolygon)
    ON_WM_KEYDOWN()
    ON_WM_CREATE()
    ON_WM_ERASEBKGND()
    ON_COMMAND(ID_OPTIONS_PREFERENCES, &CTriangleView::
        OnOptionsPreferences)
    ON_COMMAND_RANGE(CTriangleView::menuBoundaryid-1,
        CTriangleView::menuBoundaryid+30,CTriangleView::
        OnBoundaryType)
    ON_UPDATE_COMMAND_UI_RANGE(CTriangleView::
        menuBoundaryid,CTriangleView::menuBoundaryid+30,
        CTriangleView::OnUpdateBoundaryType)
    ON_WM_CONTEXTMENU()
    ON_COMMAND(ID_FILE_SOLVE, &CTriangleView::OnFileSolve)
    ON_WM_MOUSEWHEEL()
    ON_COMMAND(ID_OPTIONS_TIMESTEP, &CTriangleView::
        OnOptionsTimestep)
END_MESSAGE_MAP()

// CTriangleView construction/destruction

CTriangleView::CTriangleView():nMode(0),nPolySides(5),
brVertex(RGB(0,0,0)),brActiveVertex(RGB(255,0,0)),
    brMidpoint(RGB(0,40,255)),clrLine(RGB(255,0,255)),
    clrActiveLine(RGB(0,255,125)),clrVertex(RGB(0,0,0)),
    clrActiveVertex(RGB(255,0,0)),
rcBounds(-10000,-10000,10000,10000),clrMidpoint(RGB
    (0,40,255))
, m_Left(0)
, m_Top(100)
, m_Right(100)
, m_Bottom(0)
,m_bAnisotropic(false)
{
}

CTriangleView::~CTriangleView()
{
}



BOOL CTriangleView::PreCreateWindow(CREATESTRUCT& cs)
{
    return CView::PreCreateWindow(cs);
}
BOOL CTriangleView::UpdateCurrentVertex( gPOINT point )
{
    CTriangleDoc* pDoc=GetDocument();
    if(pDoc->m_bTri)
        return FALSE;
    pDoc->domain.MoveVertex(point);
    Invalidate(FALSE);
}
namespace gFuctionClasses
{
    template<class C> class LineTo
    {
        CDC* pDC;
    public:
        LineTo(CDC* pdc):pDC(pdc){};
        void operator() (C& elem)const
        {
            pDC->LineTo(elem);
        }
    };
    class Triangle
    {
        CDC* pDC;
        vector<gPOINT>& a;
    public:
        Triangle(CDC* pdc, vector<gPOINT>& A):pDC(pdc),a(A)
            {};
        void operator() (CTriangle& elem)const
        {
            pDC->MoveTo(a[elem.i1]);
            pDC->LineTo(a[elem.i2]);
            pDC->LineTo(a[elem.i3]);
            pDC->LineTo(a[elem.i1]);

            //draws indexes
            /*CString string;
            string.Format(_T("%d"),elem.i1);
            pDC->TextOut(a[elem.i1].x-4,a[elem.i1].y-8,
                string);
            string.Format(_T("%d"),elem.i2);
            pDC->TextOut(a[elem.i2].x-4,a[elem.i2].y-8,
                string);
            string.Format(_T("%d"),elem.i3);
            pDC->TextOut(a[elem.i3].x-4,a[elem.i3].y-8,
                string);*/
        }
    };
}
static unsigned char rainbow[780][3]={{130,0,0}, {131,0,0},
    {132,0,0}, {133,0,0}, {134,0,0}, {135,0,0}, {136,0,0},
   {138,0,0}, {140,0,0}, {141,0,0}, {142,0,0}, {143,0,0},
   {144,0,0}, {145,0,0}, {147,0,0}, {148,0,0}, {150,0,0},
   {151,0,0}, {153,0,0}, {154,0,0}, {155,0,0}, {156,0,0},
   {158,0,0}, {159,0,0}, {161,0,0}, {162,0,0}, {163,0,0},
   {164,0,0}, {166,0,0}, {167,0,0}, {168,0,0}, {169,0,0},
   {171,0,0}, {172,0,0}, {173,0,0}, {175,0,0}, {176,0,0},
   {177,0,0}, {178,0,0}, {179,0,0}, {182,0,0}, {183,0,0},
   {184,0,0}, {185,0,0}, {187,0,0}, {188,0,0}, {189,0,0},
   {190,0,0}, {192,0,0}, {194,0,0}, {195,0,0}, {196,0,0},
   {197,0,0}, {198,0,0}, {199,0,0}, {200,0,0}, {203,0,0},
   {204,0,0}, {205,0,0}, {206,0,0}, {207,0,0}, {208,0,0},
   {209,0,0}, {212,0,0}, {213,0,0}, {214,0,0}, {216,0,0},
   {217,0,0}, {218,0,0}, {219,0,0}, {221,0,0}, {223,0,0},
   {224,0,0}, {225,0,0}, {226,0,0}, {227,0,0}, {228,0,0},
   {229,0,0}, {231,0,0}, {233,0,0}, {234,0,0}, {235,0,0},
   {236,0,0}, {237,0,0}, {238,0,0}, {240,0,0}, {241,0,0},
   {243,0,0}, {244,0,0}, {246,0,0}, {247,0,0}, {248,0,0},
   {249,0,0}, {251,0,0}, {252,0,0}, {252,1,0}, {253,2,0},
{253,3,0}, {253,4,0}, {254,4,0}, {254,5,0}, {254,6,0},
{255,7,0}, {255,9,0}, {255,10,0}, {255,11,0},
{255,13,0}, {255,14,0}, {255,15,0}, {255,16,0},
{255,17,0}, {255,20,0}, {255,21,0}, {255,22,0},
{255,23,0}, {255,25,0}, {255,26,0}, {255,27,0},
{255,28,0}, {255,30,0}, {255,32,0}, {255,33,0},
{255,34,0}, {255,35,0}, {255,36,0}, {255,37,0},
{255,38,0}, {255,41,0}, {255,42,0}, {255,43,0},
{255,44,0}, {255,45,0}, {255,46,0}, {255,47,0},
{255,50,0}, {255,51,0}, {255,52,0}, {255,54,0},
{255,55,0}, {255,56,0}, {255,57,0}, {255,59,0},
{255,61,0}, {255,62,0}, {255,63,0}, {255,64,0},
{255,65,0}, {255,66,0}, {255,67,0}, {255,69,0},
{255,71,0}, {255,72,0}, {255,73,0}, {255,74,0},
{255,75,0}, {255,76,0}, {255,78,0}, {255,79,0},
{255,81,0}, {255,82,0}, {255,84,0}, {255,85,0},
{255,86,0}, {255,87,0}, {255,89,0}, {255,90,0},
{255,92,0}, {255,93,0}, {255,94,0}, {255,95,0},
{255,97,0}, {255,98,0}, {255,99,0}, {255,100,0},
{255,102,0}, {255,103,0}, {255,104,0}, {255,106,0},
{255,107,0}, {255,108,0}, {255,109,0}, {255,110,0},
{255,113,0}, {255,114,0}, {255,115,0}, {255,116,0},
{255,118,0}, {255,119,0}, {255,120,0}, {255,121,0},
{255,123,0}, {255,124,0}, {255,125,0}, {255,126,0},
{255,127,0}, {255,128,0}, {255,129,0}, {255,130,0},
{255,133,0}, {255,134,0}, {255,135,0}, {255,136,0},
{255,137,0}, {255,138,0}, {255,139,0}, {255,142,0},
{255,143,0}, {255,144,0}, {255,146,0}, {255,147,0},
{255,148,0}, {255,149,0}, {255,151,0}, {255,153,0},
{255,154,0}, {255,155,0}, {255,156,0}, {255,157,0},
{255,158,0}, {255,159,0}, {255,161,0}, {255,163,0},
{255,164,0}, {255,165,0}, {255,166,0}, {255,167,0},
{255,168,0}, {255,170,0}, {255,171,0}, {255,173,0},
{255,174,0}, {255,176,0}, {255,177,0}, {255,178,0},
{255,179,0}, {255,181,0}, {255,182,0}, {255,184,0},
{255,185,0}, {255,186,0}, {255,187,0}, {255,189,0},
{255,190,0}, {255,191,0}, {255,192,0}, {255,194,0},
{255,195,0}, {255,196,0}, {255,198,0}, {255,199,0},
{255,200,0}, {255,201,0}, {255,202,0}, {255,205,0},
{255,206,0}, {255,207,0}, {255,208,0}, {255,210,0},
{255,211,0}, {255,212,0}, {255,213,0}, {255,215,0},
{255,217,0}, {255,218,0}, {255,219,0}, {255,220,0},
{255,221,0}, {255,222,0}, {255,223,0}, {255,226,0},
{255,227,0}, {255,228,0}, {255,230,0}, {255,231,0},
{255,232,0}, {255,233,0}, {255,236,0}, {255,237,0},
{255,238,0}, {255,239,0}, {255,240,0}, {255,241,0},
{255,242,0}, {255,244,0}, {255,246,0}, {255,247,0},
{255,248,0}, {255,249,0}, {255,250,0}, {255,251,0},
{255,252,0}, {255,254,0}, {252,254,2}, {251,254,3},
{250,254,4}, {249,254,5}, {248,254,6}, {247,254,7},
{246,254,9}, {246,255,10}, {243,255,12}, {242,255,13},
{241,255,14}, {240,255,15}, {239,255,16}, {237,255,18},
{236,255,19}, {236,255,20}, {233,255,22}, {232,255,23},
{231,255,24}, {230,255,25}, {228,255,27}, {227,255,28},
{226,255,29}, {226,255,30}, {222,255,33}, {221,255,34},
{220,255,35}, {219,255,36}, {218,255,37}, {216,255,39},
{215,255,40}, {215,255,41}, {212,255,43}, {211,255,44},
{209,255,46}, {208,255,47}, {207,255,48}, {206,255,49},
{205,255,50}, {205,255,51}, {202,255,53}, {200,255,55},
{199,255,56}, {198,255,57}, {197,255,58}, {196,255,59},
{195,255,60}, {195,255,61}, {191,255,64}, {190,255,65},
{189,255,66}, {187,255,68}, {186,255,69}, {185,255,70},
{184,255,71}, {181,255,74}, {180,255,75}, {179,255,76},
{178,255,77}, {177,255,78}, {176,255,79}, {175,255,80},
{174,255,82}, {171,255,84}, {170,255,85}, {169,255,86},
{168,255,87}, {167,255,88}, {166,255,89}, {165,255,90},
{164,255,92}, {161,255,94}, {160,255,95}, {158,255,97},
{157,255,98}, {156,255,99}, {155,255,100},
{153,255,102}, {153,255,103}, {150,255,105},
{149,255,106}, {148,255,107}, {147,255,108},
{146,255,109}, {144,255,111}, {143,255,112},
{143,255,113}, {140,255,115}, {139,255,116},
{138,255,117}, {137,255,118}, {135,255,120},
{134,255,121}, {133,255,122}, {133,255,123},
{130,255,125}, {129,255,126}, {128,255,128},
{126,255,129}, {125,255,130}, {124,255,131},
{123,255,132}, {123,255,133}, {120,255,135},
{119,255,136}, {117,255,138}, {116,255,139},
{115,255,140}, {114,255,141}, {113,255,142},
{113,255,143}, {110,255,145}, {108,255,147},
{107,255,148}, {106,255,149}, {105,255,150},
{104,255,151}, {103,255,152}, {103,255,153},
{99,255,156}, {98,255,157}, {97,255,158}, {95,255,160},
{94,255,161}, {93,255,162}, {92,255,163}, {89,255,166},
{88,255,167}, {87,255,168}, {86,255,169}, {85,255,170},
{84,255,171}, {83,255,172}, {82,255,174}, {79,255,176},
{78,255,177}, {77,255,178}, {76,255,179}, {75,255,180},
{74,255,181}, {73,255,182}, {72,255,184}, {69,255,186},
{68,255,187}, {66,255,189}, {65,255,190}, {64,255,191},
{63,255,192}, {61,255,194}, {61,255,195}, {58,255,197},
{57,255,198}, {56,255,199}, {55,255,200}, {54,255,201},
{52,255,203}, {51,255,204}, {51,255,205}, {48,255,207},
{47,255,208}, {46,255,209}, {45,255,210}, {43,255,212},
{42,255,213}, {41,255,214}, {41,255,215}, {37,255,218},
{36,255,219}, {35,255,220}, {34,255,221}, {33,255,222},
{31,255,224}, {30,255,225}, {30,255,226}, {27,255,228},
{26,255,229}, {24,255,231}, {23,255,232}, {22,255,233},
{21,255,234}, {20,255,235}, {20,255,236}, {17,255,238},
{15,255,240}, {14,255,241}, {13,255,242}, {12,255,243},
{11,255,244}, {10,255,245}, {10,255,246}, {6,254,248},
{5,254,249}, {4,254,250}, {3,254,251}, {2,254,252},
{1,254,253}, {0,254,254}, {0,251,255}, {0,250,255},
{0,249,255}, {0,248,255}, {0,247,255}, {0,246,255},
{0,245,255}, {0,244,255}, {0,241,255}, {0,240,255},
{0,239,255}, {0,238,255}, {0,237,255}, {0,236,255},
{0,235,255}, {0,234,255}, {0,231,255}, {0,230,255},
{0,228,255}, {0,227,255}, {0,226,255}, {0,225,255},
{0,223,255}, {0,223,255}, {0,220,255}, {0,219,255},
{0,218,255}, {0,217,255}, {0,216,255}, {0,214,255},
{0,213,255}, {0,213,255}, {0,210,255}, {0,208,255},
{0,207,255}, {0,206,255}, {0,205,255}, {0,203,255},
{0,202,255}, {0,202,255}, {0,199,255}, {0,198,255},
{0,197,255}, {0,195,255}, {0,194,255}, {0,193,255},
{0,192,255}, {0,192,255}, {0,189,255}, {0,188,255},
{0,186,255}, {0,185,255}, {0,184,255}, {0,183,255},
{0,182,255}, {0,182,255}, {0,178,255}, {0,177,255},
{0,176,255}, {0,174,255}, {0,173,255}, {0,172,255},
{0,171,255}, {0,171,255}, {0,167,255}, {0,166,255},
{0,165,255}, {0,164,255}, {0,163,255}, {0,162,255},
{0,161,255}, {0,158,255}, {0,157,255}, {0,156,255},
{0,155,255}, {0,154,255}, {0,153,255}, {0,152,255},
{0,151,255}, {0,148,255}, {0,147,255}, {0,146,255},
{0,144,255}, {0,143,255}, {0,142,255}, {0,141,255},
{0,140,255}, {0,137,255}, {0,136,255}, {0,135,255},
{0,134,255}, {0,133,255}, {0,132,255}, {0,130,255},
{0,130,255}, {0,127,255}, {0,126,255}, {0,125,255},
{0,124,255}, {0,123,255}, {0,122,255}, {0,121,255},
{0,121,255}, {0,118,255}, {0,116,255}, {0,115,255},
{0,114,255}, {0,113,255}, {0,111,255}, {0,110,255},
{0,110,255}, {0,107,255}, {0,106,255}, {0,105,255},
{0,103,255}, {0,102,255}, {0,101,255}, {0,100,255},
{0,100,255}, {0,97,255}, {0,96,255}, {0,94,255},
{0,93,255}, {0,92,255}, {0,91,255}, {0,90,255},
{0,90,255}, {0,86,255}, {0,85,255}, {0,84,255},
{0,82,255}, {0,81,255}, {0,80,255}, {0,79,255},
{0,79,255}, {0,75,255}, {0,74,255}, {0,73,255},
{0,72,255}, {0,71,255}, {0,70,255}, {0,69,255},
{0,66,255}, {0,65,255}, {0,64,255}, {0,63,255},
{0,62,255}, {0,61,255}, {0,60,255}, {0,59,255},
{0,56,255}, {0,55,255}, {0,54,255}, {0,52,255},
{0,51,255}, {0,50,255}, {0,49,255}, {0,48,255},
{0,45,255}, {0,44,255}, {0,43,255}, {0,42,255},
{0,41,255}, {0,40,255}, {0,38,255}, {0,38,255},
{0,35,255}, {0,34,255}, {0,33,255}, {0,32,255},
{0,31,255}, {0,29,255}, {0,28,255}, {0,28,255},
{0,25,255}, {0,23,255}, {0,22,255}, {0,21,255},
{0,20,255}, {0,18,255}, {0,17,255}, {0,17,255},
    {0,14,255}, {0,13,255}, {0,12,255}, {0,10,255},
    {0,9,255}, {0,8,255}, {0,7,255}, {0,7,255}, {0,5,254},
    {0,4,253}, {0,3,253}, {0,2,253}, {0,1,252}, {0,1,252},
    {0,0,252}, {0,0,252}, {0,0,248}, {0,0,247}, {0,0,246},
    {0,0,244}, {0,0,243}, {0,0,242}, {0,0,241}, {0,0,241},
    {0,0,237}, {0,0,236}, {0,0,235}, {0,0,234}, {0,0,233},
    {0,0,232}, {0,0,231}, {0,0,228}, {0,0,227}, {0,0,226},
    {0,0,225}, {0,0,224}, {0,0,223}, {0,0,222}, {0,0,221},
    {0,0,218}, {0,0,217}, {0,0,216}, {0,0,214}, {0,0,213},
    {0,0,212}, {0,0,211}, {0,0,210}, {0,0,207}, {0,0,206},
    {0,0,205}, {0,0,204}, {0,0,203}, {0,0,202}, {0,0,200},
    {0,0,200}, {0,0,197}, {0,0,196}, {0,0,195}, {0,0,194},
    {0,0,193}, {0,0,191}, {0,0,190}, {0,0,190}, {0,0,187},
    {0,0,185}, {0,0,184}, {0,0,183}, {0,0,182}, {0,0,180},
    {0,0,179}, {0,0,179}, {0,0,176}, {0,0,175}, {0,0,174},
    {0,0,172}, {0,0,171}, {0,0,170}, {0,0,169}, {0,0,169},
    {0,0,166}, {0,0,165}, {0,0,163}, {0,0,162}, {0,0,161},
    {0,0,160}, {0,0,159}, {0,0,159}, {0,0,155}, {0,0,154},
    {0,0,153}, {0,0,151}, {0,0,150}, {0,0,149}, {0,0,148},
    {0,0,148}, {0,0,144}, {0,0,143}, {0,0,142}, {0,0,141},
    {0,0,140}, {0,0,139}, {0,0,138}, {0,0,135}, {0,0,134},
    {0,0,133}, {0,0,132}, {0,0,131}, {0,0,130}, {0,0,129} };
COLORREF FloatToRainBow(float v)
{
    float t=(780-v*780);//(271-v*272);
    int i=(t>0)?t:0;
    return RGB(rainbow[i][0],rainbow[i][1],rainbow[i][2]);
}
int FloatToRainBowI(float v)
{
    v=abs(v);
    float t=(779-v*779);//(271-v*272);
    int i=(t>0)?t:0;
    return i;
}
void CTriangleView::Spline(CDC* pDC,gPOINT p1,gPOINT p2)
{
    /*pDC->MoveTo(RT(p2));
    for(double k=0.1;k<1;k+=0.1)
        pDC->LineTo(RT(gPOINT(k*k*p1.x+2*k*(1-k)*p1.mx+(1-k
            )*(1-k)*p2.x,
                                k*k*p1.y+2*k*(1-k)*p1.my+(1-k
                                    )*(1-k)*p2.y)));*/
}
void CTriangleView::OnDraw(CDC* pDC1)
{
    CTriangleDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
```

```
        return;
    //...switch to back buffer
    CDC DC;DC.CreateCompatibleDC(pDC1);
    CDC* pDC=&DC;
    CBitmap* oldBitmap=pDC->SelectObject(&bmpBuffer);
    //...clear everything
    CRect rect;GetClientRect(&rect);
    pDC->FillRect(&rect,&CBrush(RGB(255,255,255)));

    if(pDoc->m_bSolved)
    {
        //pDC1->FillRect(&rect,&CBrush(RGB(255,255,255)));
        //determine min and max
        double min,max;
        double* u=pDoc->u;
        min=max=u[0];
        int n=pDoc->m_ptVector.size();
        for(int i=0;i<n;i++)
            if(u[i]>max)
                max=u[i];
            else if(u[i]<min)
                min=u[i];
        //...
        TRIVERTEX vert[3];
        GRADIENT_TRIANGLE gTri[1];
        gTri[0].Vertex1   = 0;
        gTri[0].Vertex2   = 1;
        gTri[0].Vertex3   = 2;
        memset(&vert,0,sizeof(vert));
        for(list<CTriangle>::iterator i=pDoc->m_triList.
            begin();i!=pDoc->m_triList.end();i++)
        {
            CPoint p1=RT(pDoc->m_ptVector[(*i).i1]);
            CPoint p2=RT(pDoc->m_ptVector[(*i).i2]);
            CPoint p3=RT(pDoc->m_ptVector[(*i).i3]);
            int ind;
            //...
            vert[0].x=p1.x;vert[0].y=p1.y;
            ind=FloatToRainBowI((u[(*i).i1]-min)/(max-min))
                ;
            vert[0].Red=rainbow[ind][0];
            vert[0].Green=rainbow[ind][1];
            vert[0].Blue=rainbow[ind][2];
            //...
            vert[1].x=p2.x;vert[1].y=p2.y;
            ind=FloatToRainBowI((u[(*i).i2]-min)/(max-min))
                ;
            vert[1].Red=rainbow[ind][0];
            vert[1].Green=rainbow[ind][1];
```

```
                vert[1].Blue=rainbow[ind][2];
                //...
                vert[2].x=p3.x;vert[2].y=p3.y;
                ind=FloatToRainBowI((u[(*i).i3]-min)/(max-min))
                    ;
                vert[2].Red=rainbow[ind][0];
                vert[2].Green=rainbow[ind][1];
                vert[2].Blue=rainbow[ind][2];
                for(int j=0;j<3;j++)
                {
                    vert[j].Red*=255;
                    vert[j].Green*=255;
                    vert[j].Blue*=255;
                }
                //...
                pDC->GradientFill(vert,3,&gTri,1,
                    GRADIENT_FILL_TRIANGLE);
            }
            DrawScale(pDC,CPoint(rect.right-85,rect.top+20),
                CSize(40,rect.Height()-100));
        }
        else
        if(pDoc->m_bTri==false)//not triangulated
        {
            pDoc->domain.Draw(pDC,this);
        }
        else//draw triangles
        {
            vector<gPOINT>&a=pDoc->m_ptVector;
            CPen pen;pen.CreatePen(PS_SOLID,1,RGB(255,0,0));
            CPen*pOldPen=pDC->SelectObject(&pen);
            vector<gPOINT> v(a.size());
            for(int i=0;i<v.size();i++)
                v[i]=RT(a[i]);

            pDC->SetBkMode(TRANSPARENT);
            LOGFONT lf;memset(&lf,0,sizeof(lf));
            lstrcpy(lf.lfFaceName,_T("Arial Narrow"));
            lf.lfHeight=14;
            CFont font;
            font.CreateFontIndirect(&lf);
            CFont* pOldFont=pDC->SelectObject(&font);
            for_each(pDoc->m_triList.begin(),pDoc->m_triList.
                end(),gFuctionClasses::Triangle(pDC,v));
            pDC->SelectObject(pOldFont);
            pDC->SelectObject(pOldPen);
            if(pDoc->m_bSolved)
            {
                double min,max;
                double* u=pDoc->u;
                min=max=u[0];
                int n=pDoc->m_ptVector.size();
                for(int i=0;i<n;i++)
                     if(u[i]>max)
                          max=u[i];
                     else if(u[i]<min)
                          min=u[i];
                CString s;
                s.Format(_T("min=%f,max=%f"),min,max);
                MessageBox(s);
                for(int i=0;i<n;i++)
                {
                     CBrush brush(FloatToRainBow((u[i]-min)/(max
                         -min)));
                     CPoint p=RT(pDoc->m_ptVector[i]);
                     pDC->FillRect(&CRect(p.x-4,p.y-4,p.x+4,p.y
                         +4),&brush);
                }
           }
     }
     BOOL r=pDC1->BitBlt(0,0,rect.Width(),rect.Height(),pDC
          ,0,0,SRCCOPY);
     pDC->SelectObject(oldBitmap);
     ReleaseDC(pDC);
}

// CTriangleView message handlers
void CTriangleView::OnInsertPolygon()
{
     nMode=VIEW_MODE_POLY_INSERT;
}

void CTriangleView::OnUpdateInsertPolygon(CCmdUI *pCmdUI)
{
     pCmdUI->SetCheck(nMode!=0);
     pCmdUI->Enable(nMode==0);
}

void CTriangleView::OnMouseMove(UINT nFlags, CPoint p)
{
     gPOINT point=Transform(p);
     if(nMode==VIEW_MODE_POLY_DRAG)
     {
          /*CTriangleDoc* pDoc=GetDocument();
          int nSize=pDoc->m_ptList.size();
          double l=length(ptCenter,point);
          list<gPOINT>&a=pDoc->m_ptList;
          for(li i=a.begin();i!=a.end();i++)
```

```
                *i=gPOINT(ptCenter.x+((*i).x-ptCenter.x)/length
                    ((*i),ptCenter)*l,
                    ptCenter.y+((*i).y-ptCenter.y)/length((*i),
                        ptCenter)*l);
            Invalidate(FALSE);*/
        }
        else
            if (nMode==VIEW_MODE_VERTEX_DRAG)
            {
                AfxGetMainWnd()->SendMessage(
                    WM_UPDATEPOINTTRACKER,0,(LPARAM)&gPOINT(
                    point));
                GetDocument()->domain.MoveVertex(point);
                Invalidate(FALSE);
            }
            else
                if(nMode==VIEW_MODE_MIDPOINT_DRAG)
                {
                    GetDocument()->domain.SetMidpoint(point);
                    Invalidate(FALSE);
                }
}

COLORREF GetPixelBox(CDC* pDC,CPoint p)
{
    static COLORREF white=RGB(255,255,255);
    COLORREF colors[5];
    int count[5]={1,0,0,0,0};
    int ci=0;
    colors[ci++]=pDC->GetPixel(p);
    CPoint points[8]={CPoint(p.x,p.y+1),CPoint(p.x,p.y-1),
                        CPoint(p.x+1,p.y+1),CPoint(p.x+1,p.y
                            -1),CPoint(p.x+1,p.y),
                        CPoint(p.x-1,p.y+1),CPoint(p.x-1,p.y
                            -1),CPoint(p.x-1,p.y)};
    for(int i=0;i<8;i++)
    {
        COLORREF c=pDC->GetPixel(points[i]);
        int j=0;
        for(;j<ci;j++)
            if(colors[j]==c)
            {
                count[j]++;
                break;
            }
        if(j==ci)
            colors[ci++]=c;
    }
    int m1=count[0],m2=count[1];
    int i1=0,i2=1;
    if(m1<m2){swap(m1,m2);swap(i1,i2);}
    for(int i=2;i<5;i++)
        if(count[i]>m1)
        {
            m2=m1;        i2=i1;
            m1=count[i];i1=i;
        }
    if(colors[i1]!=white)
        return colors[i1];
    else
        return colors[i2];
}

void CTriangleView::OnLButtonDown(UINT nFlags, CPoint p)
{
    SetCapture();
    gPOINT point=Transform(p);
    CTriangleDoc* pDoc=GetDocument();
    if(pDoc->m_bSolved)
    {
        AfxGetMainWnd()->SendMessage(WM_UPDATEPOINTTRACKER
            ,0,(LPARAM)&point);
        double v=pDoc->Eval(point.x,point.y);
        if(v==10e20)
            return;
        AfxGetMainWnd()->SendMessage(WM_UPDATEPOINTTRACKER
            ,1,(LPARAM)&v);
    }
    if(pDoc->m_bTri)
        return;
    if(nMode==VIEW_MODE_POLY_INSERT)
    {
        //nPolySides=((CMainFrame*)AfxGetMainWnd())->
            GetPolySides();
        //for(int i=0;i<nPolySides;i++)
        //  pDoc->m_ptList.push_back(gPOINT(point.x+10*sin
            (/*PI/4+*/i*PI2/nPolySides),point.y+10*cos(/*PI
            /4+*/i*PI2/nPolySides)));
        //currVertex=--(pDoc->m_ptList.end());
        //nMode=VIEW_MODE_POLY_DRAG;
        //ptCenter=CPoint(point.x,point.y);
        //Invalidate(FALSE);
    }
    else
    {
        CDC* pDC=GetDC();
        AfxGetMainWnd()->SendMessage(WM_UPDATEPOINTTRACKER
            ,0,(LPARAM)&point);
        if((pDC->GetPixel(p)==clrMidpoint))//clicked a
           midpoint - start dragging
        {
            nMode=VIEW_MODE_MIDPOINT_DRAG;
            Invalidate(FALSE);
        }else

        if((pDC->GetPixel(p)==clrVertex)||(pDC->GetPixel(p)
           ==clrActiveVertex))//clicked a vertex - start
           dragging
        {
            pDoc->domain.SelectVertexAt(point);
            nMode=VIEW_MODE_VERTEX_DRAG;
            Invalidate(FALSE);
        }
        else
            if(GetPixelBox(pDC,p)==clrLine)//clicked a line
                - select vertex to which it corresponds
            {
                pDoc->domain.SelectCurveAt(point);
                Invalidate(FALSE);
            }
            else//add new vertex
            {
                pDoc->domain.AddVertex(point);
                Invalidate(FALSE);
            }
        ReleaseDC(pDC);
    }
}

void CTriangleView::OnLButtonUp(UINT nFlags, CPoint point)
{
    ReleaseCapture();
    CTriangleDoc* pDoc=GetDocument();
    if(pDoc->m_bTri)
        return;
    nMode=0;
    Invalidate(FALSE);
}
void CTriangleView::OnFileTrianglulate()
{
    CTriangleDoc* pDoc=GetDocument();
    if(pDoc->m_bTri==false)
    {
        //Triangulation T(&pDoc->m_ptList);
        Triangulation T(&pDoc->domain);
        T.triangulate(pDoc->m_Area);
```

```
            pDoc->m_ptVector=T.points;
            pDoc->m_triList=T.triangles;
            pDoc->m_segList=T.segments;

            pDoc->m_bTri=true;
            CTriangle::v=pDoc->m_ptVector;
            Invalidate(FALSE);
        }
        else
        {
            pDoc->m_triList.clear();
            pDoc->m_segList.clear();
            pDoc->m_bTri=false;
            pDoc->m_bSolved=false;
            Invalidate(FALSE);
        }
        pDoc->UpdateAllViews(0);
}

void CTriangleView::OnKeyDown(UINT nChar, UINT nRepCnt,
    UINT nFlags)
{
        CTriangleDoc* pDoc=GetDocument();
        switch(nChar)
        {
        case VK_DELETE://remove current vertex
            {
                if(pDoc->m_bTri)
                    return;
                pDoc->domain.DeleteVertex();
                Invalidate(FALSE);
            }break;
        case VK_LEFT:
        case VK_DOWN:
            {
                pDoc->domain.NextVertex();
                Invalidate(FALSE);
            }break;
        case VK_RIGHT:
        case VK_UP:
            {
                pDoc->domain.PrevVertex();
                Invalidate(FALSE);
            }break;
        default:
            CView::OnKeyDown(nChar, nRepCnt, nFlags);
        }
}
```

```
// CTriangleView diagnostics

#ifdef _DEBUG
void CTriangleView::AssertValid() const
{
    CView::AssertValid();
}

void CTriangleView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CTriangleDoc* CTriangleView::GetDocument() const // non-
   debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTriangleDoc
        )));
    return (CTriangleDoc*)m_pDocument;
}
#endif //_DEBUG
#include "triangle1.h"



int CTriangleView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    CRect rect;GetClientRect(&rect);
    bmpBuffer.CreateBitmap(1280,1024,1,32,0);
    //HBITMAP bitmap=CreateBitmap(rect.Width(),rect.Height
        (),1,24,0);
    //iImage.Attach(bitmap);
    //iImage.Create(rect.Width(),rect.Height(),32);
    //int e=GetLastError();
    //iImage.LoadFromResource(AfxGetInstanceHandle(),
        IDB_TESTBITMAP);
    LOGFONT lf;memset(&lf,0,sizeof(lf));
    lf.lfHeight=15;
    lstrcpy(lf.lfFaceName,_T("Arial"));
    fontMinmax.CreateFontIndirect(&lf);
    return 0;
}

BOOL CTriangleView::OnEraseBkgnd(CDC* pDC)
{
    CView::OnEraseBkgnd(pDC);
    return 0;
}

void CTriangleView::OnInitialUpdate()
{
    CView::OnInitialUpdate();
    CTriangleDoc* pDoc=GetDocument();
    m_Left=pDoc->m_Left;
    m_Top=pDoc->m_Top;
    m_Right=pDoc->m_Right;
    m_Bottom=pDoc->m_Bottom;

}

void CTriangleView::OnOptionsPreferences()
{
    CTriangleDoc* pDoc=GetDocument();
    CPrefDialog dlg;
    dlg.m_Left=m_Left=pDoc->m_Left;
    dlg.m_Top=m_Top=pDoc->m_Top;
    dlg.m_Right=m_Right=pDoc->m_Right;
    dlg.m_Bottom=m_Bottom=pDoc->m_Bottom;
    dlg.m_Area=pDoc->m_Area;
    dlg.m_bAnisotropic=m_bAnisotropic;
    dlg.m_bLabels=pDoc->m_bBoundaryLabels;
    if(dlg.DoModal()==IDOK)
    {
        if(pDoc->m_bTri)
            OnFileTrianglulate();
        pDoc->m_Left=m_Left=m_Left=dlg.m_Left;
        pDoc->m_Top=m_Top=dlg.m_Top;
        pDoc->m_Right=m_Right=dlg.m_Right;
        pDoc->m_Bottom=m_Bottom=dlg.m_Bottom;
        pDoc->m_Area=dlg.m_Area;
        m_bAnisotropic=dlg.m_bAnisotropic;
        pDoc->m_bBoundaryLabels=dlg.m_bLabels;

        Invalidate(FALSE);
    }
}

void CTriangleView::OnBoundaryType(UINT nID)
{
    if(nID==menuCurvedid)
    {
        GetDocument()->domain.ToggleCurved();
        Invalidate(FALSE);
        return;
    }
    GetDocument()->domain.SetBorder(nID-menuBoundaryid);
}

void CTriangleView::OnUpdateBoundaryType(CCmdUI* pCmdUI)
{
    pCmdUI->SetRadio(GetDocument()->domain.GetBorder()==(
        pCmdUI->m_nID-menuBoundaryid));
}

void CTriangleView::OnContextMenu(CWnd* pWnd, CPoint point)
{
    CTriangleDoc* pDoc=GetDocument();
    if( /*(pDoc->m_ptList.size()<3)||*/(pDoc->m_bTri) )
        return;
    CMenu menu;
    menu.CreatePopupMenu();
    int id=menuBoundaryid;
    wchar_t string[50];
    int n=pDoc->m_bndVector.size();
    for(int i=0;i<n;i++)
    {
        wsprintf(string,L"boundary %i",i);
        menu.AppendMenu(CF_TEXT,id+i,string);
    }
    if(pDoc->domain.GetCurved())
        wsprintf(string,L"make straight");
    else
        wsprintf(string,L"make curved");
    menu.AppendMenu(CF_TEXT,menuCurvedid,string);

    menu.TrackPopupMenu(TPM_TOPALIGN,point.x,point.y,
        AfxGetMainWnd());
    CWnd::OnContextMenu(pWnd,point);
}

void CTriangleView::OnFileSolve()
{
    CTriangleDoc* pDoc=GetDocument();
    if(pDoc->m_bTri==false)
        OnFileTrianglulate();
    minmax=pDoc->Solve();
    Invalidate(FALSE);
}

void CTriangleView::DrawScale( CDC*pDC,CPoint point,CSize
    size)
{
    const int steps=size.cy;
    const double h=1.0/steps;
    double v=1;
    CPoint ptStart=point;
    CPoint ptEnd=ptStart+CSize(size.cx,0);
    for(int i=0;i<steps;i++)
    {
        CPen pen(PS_SOLID,1,FloatToRainBow(v));
        pDC->SelectObject(&pen);
        pDC->MoveTo(ptStart);ptStart.y+=1;
        pDC->LineTo(ptEnd);ptEnd.y+=1;
        v-=h;
        pDC->SelectStockObject(NULL_PEN);
    }
    //pDC->SetBkMode(TRANSPARENT);
    CBrush brWhite(RGB(255,255,255));
    CFont*pOldFont=pDC->SelectObject(&fontMinmax);
    CString s;
    point.y-=17;
    pDC->FillRect(&CRect(point,CSize(100,15)),&brWhite);
    s.Format(_T("%g"),minmax.x);
    pDC->TextOut(point.x,point.y/*-17*/,s);
    s.Format(_T("%g"),minmax.y);
    pDC->FillRect(&CRect(ptStart,CSize(100,15)),&brWhite);
    pDC->TextOut(ptStart.x,ptStart.y,s);
    pDC->SelectObject(pOldFont);

}
BOOL CTriangleView::OnMouseWheel(UINT nFlags, short zDelta,
     CPoint pt)
{
    CTriangleDoc*pDoc=GetDocument();
    if(zDelta<0)
        pDoc->domain.NextVertex();
    else
    if(zDelta>0)
        pDoc->domain.PrevVertex();
    Invalidate(FALSE);
    return CView::OnMouseWheel(nFlags, zDelta, pt);
}

//
void CTriangleView::OnOptionsTimestep()
{
}

void CTriangleView::OnUpdate(CView* pSender, LPARAM lHint,
    CObject* pHint)
{
    Invalidate(FALSE);
}
```