Geometric Modeling
Assignment sheet #6
"Bezier Curves, kd-Tree & kNN"
(due June 13th/June 15th 2012 during the interviews)

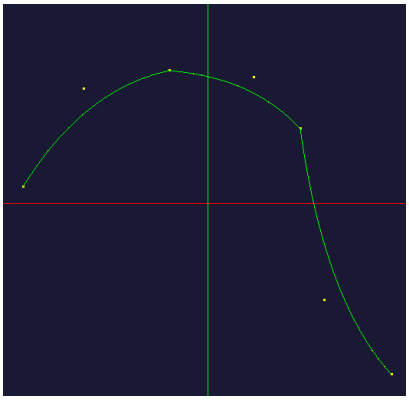
Silke Jansen, Ruxandra Lasowski,
Avinash Sharma, **Art Tevs**, Michael Wand

(1) Bezier Curve [2+2+1 points]

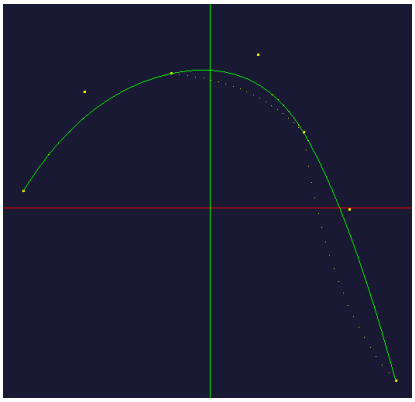
a) Create a new 2D experiment. Add several control points. Add a button that draws a *quadratic* Bezier curve specified by the control points. Draw the curve in green. Let user specify the number of line segments used to approximate each curve segment.

b) Same as (a), however now draw a *cubic* Bezier curve with red color.

c) Add a button which makes the curve from (a) C^1-continuous.



Quadratic Bezier curve



C1-continuous quadratic Bezier curve

(2) kd-Tree and kNN search [8+2+5 points]

In this exercise you will implement a kd-tree spatial data structure. The kd-tree will be used to speed-up nearest neighbor searches. Please note that the drawing exercise (b) is not required to solve (c), however helps you to debug the kd-tree you implement.

a) Add a button which builds a kd-tree for any set of specified points. The split axis alternates between *x* and *y*-axis, i.e. root node is split along x-axis, its children along y-axis, it's grandchildren along x-axis... As splitting criterion use the average, i.e. a node divides the space into two half-spaces with a border along the average of included points. The maximal allowed number of points in a node should be specified by the user (default 5).
Please note that instead of a median point split we use average here.
Hint: Building the tree recursively makes the exercise a lot easier.

b) Add a button which draws the kd-tree. You should draw the bounding box of your data points first. Every node splits the corresponding subspace into two child nodes. A split can be

represented by a simple purple line. For debugging purpose you can set the thickness of the split line according to node's depth in the tree.
Hint: Perform drawing recursively.

c) Implement *k-nearest-neighbor (kNN)* search using a brute force approach as well as with the help of the kd-tree you computed before. Select a point for which you would like to find *k* nearest neighbors. Let user specify the value of *k*. Mark all found points by changing their color to green. Compare the running times of both approaches using the *Timer* class found in "system/misc/Timer.h".
Test your implementation with huge number of points (e.g. 10000). For this consider to place points randomly on the screen in a small area (e.g. in the [-5;5] interval).
Hint: Use the patch from the previous exercise to simply access the selected point.

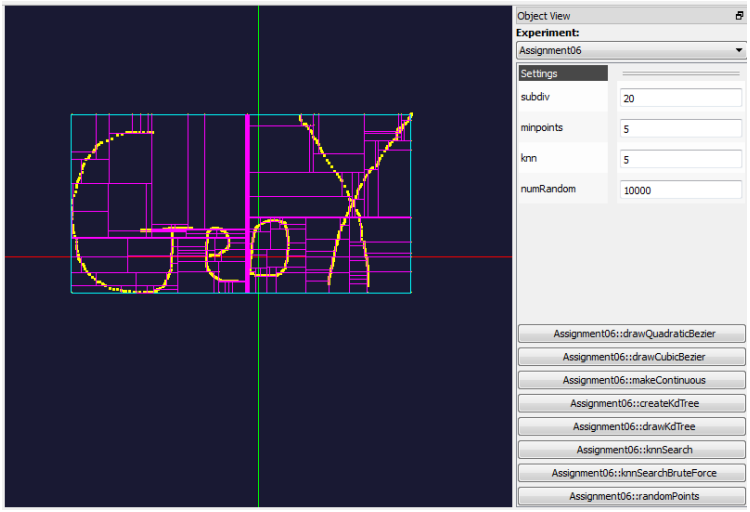
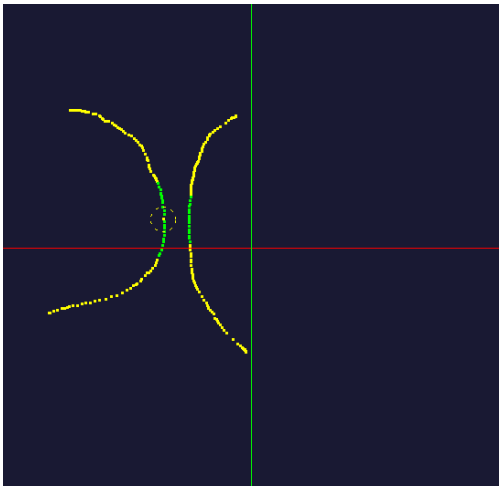
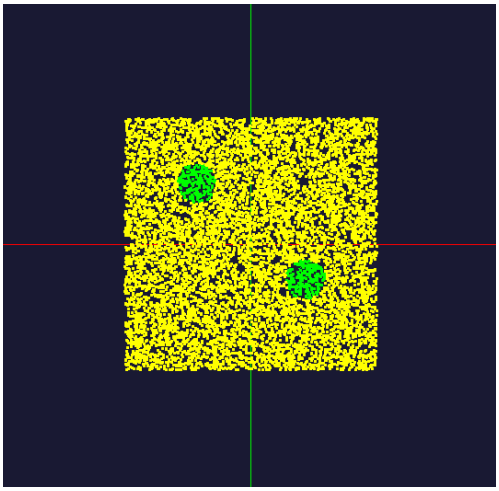


Illustration of the kd-tree



40 nearest neighbors



200 nearest neighbors in a data set containing 10000 random points. (left-top) kd-tree: 191 [ms], (right-bottom) brute force: 347 [ms]