Geometric Modeling
Assignment sheet #10
"Projection and Rational Bezier Curves"
(due July 11th/July 13th 2012 during the interviews)

Silke Jansen, Ruxandra Lasowski,
Art Tevs, Michael Wand

---

(1) "Do" projection / Rational Bezier Curves [3+4+3 points]

a. Create a new experiment. Draw a 3D cube $[-d,d]^3$ (with $0 < d \leq 1$ specified by user) centered at **x** (user should specify **x**) and projected to 2D plane by perspective projection. Assume pinhole camera model. Connect cube's corner points with lines.

b. Draw 4 quadratic Bezier curves so that their projection corresponds to a circle in 2D. Let user specify the radius of the circle. Use rational Bezier curves.
Hint: You can hard code curve parameters/weights in your code.

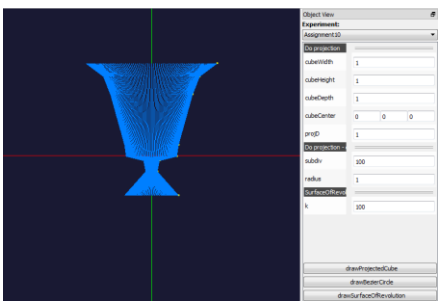
c. Create a new experiment with GLGeometryViewer3D as main viewer, i.e. 3D experiment. Draw same curves as in b) however, now in 3D space. Move camera around to see how the four Bezier curves looks like in 3D space which visualizes the homogenous space.

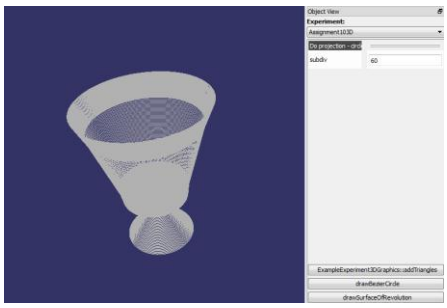
(2) Surface of Revolution [3+3 Points]

a. Create a new 2D experiment. Plot points which represents the " generatrix" for a surface of revolution. Add a button which rotates the generatrix around the y-axis and renders it *k* times for a full revolution. Don't forget to add points in between the plotted points, in order to fill the empty space (linear interpolation is enough).

b. Since surface of revolution is better viewed in 3D, you are asked in this part of the exercise to render the generatrix you created in (a) in 3D space. For this create a new 3D experiment with GLGeometryViewer3D as main viewer. Add a button which plots the generatrix around the y-axis. You might need to increase *k* to get denser point cloud. Look on the resulting object by rotating the camera around.

Hint: In order to transfer points from your 2D experiment into 3D, you can create a global object which holds the generatrix or use the singleton paradigm for your experiment (no matter if this is a bad coding practice or not☺)





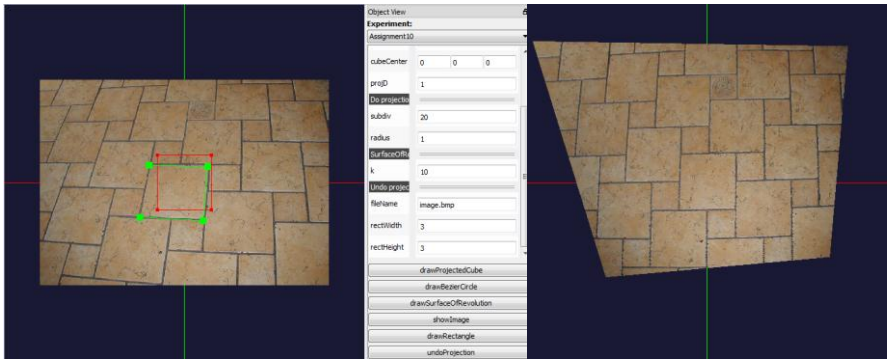
(3) BONUS: "Undo" projection [7 + 3 points]

For this exercise you need to update your GeoX package with patch provided on the lecture webpage. Patch contains the files `ImageLoader.h/cpp` which implements loads a bitmap (*.bmp) image and add it as simple 2D point cloud. Add a button which loads the image.

In this exercise you are asked to implement a method for removing projective distortion from a perspective image of a plane. This is for example useful for creating orthogonal/parallel projection image out of one taken by pinhole camera. In the following we assume that we are working with a projection of a 3D plane onto 2D image plane only. Everything is applied in 2D space.

a. A pinhole camera model is a perspective projection of a 3D point on 2D image plane. However due to perspective projection the original shape is distorted. In general parallel lines on a scene plane are not parallel and converge to a finite point in the image plane. There exists a 3x3 matrix **H**, which transforms the real projection plane (no distortion) to image plane (perspective distortion), so $\mathbf{x'}_i = \mathbf{H}\mathbf{x}_i$. **H** has capability of mapping straight lines to straight lines. Given four corresponding point pairs, i.e. $(\hat{\mathbf{x}}_1, \mathbf{x}_1)$, …, $(\hat{\mathbf{x}}_4, \mathbf{x}_4)$, where $\hat{\mathbf{x}}_i$ represents $\mathbf{x'}_i$ in homogenous coordinates, compute **H**. Assume that h_{33}=1. How would you compute **H** if h_{33} were unknown?
*Hint: **x** and $\hat{\mathbf{x}}$ are represented in homogenous coordinates, i.e. (x, y, 1).*

b. Add a button which draws a rectangle made of $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ with user specified width and height centered at origin. Load an image and select four points $(\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \hat{\mathbf{x}}_3, \hat{\mathbf{x}}_4)$ corresponding to the drawn rectangle. "Undo" the perspective projection by transforming image points with computed $\mathbf{H}^{-1}$. Experiment with different images.
Hint: Take care of the order for your points.
Hint: Use `GLGeometryViewer::setPointSelectedCallback` from the patch to access points as soon as they are selected.



(left) Loaded image as colored point cloud. The user selected rectangle is green. The target rectangle is red. (right) Result of „undo" projection.