CAD practical – programming part 5:
Subdivision schemes (Chaïkin and Doo-Sabin)

Generate a CodeBlocks project with the files from the zip file (`CADCG_W5_code.zip`) by using CMake. For this work, two different executables will be compiled: Chaikin (using `ntest/nchaikin.cc` for main function) and Doo-Sabin (using `doosabin.cc` for main function). To compile each of these new executables, use the correct Build Target: "`chaikin-static`" or "`doosabin-static`".

Chaïkin's subdivision scheme

Chaïkin's scheme is a curve subdivision algorithm. Its purpose is to obtain a smooth curve from a given number of control points forming a polygon (Theory: ppt 5 pp. 44-48).

1. Compile. In the FLTK window: the green points are the control points forming the white polygon which the Chaïkin's subdivision algorithm will be applied on. The red curve is the B-spline the subdivision algorithm must converge to.
2. Implement the function `void nchaikin::refine` (in `api/nchaikin.cc`).
   This function takes the desired number of refinement steps in argument. You must define the new control points and their number for the refined curve. The main file does not need to be modified. Tip: `nchaikin` is a child class of `curve`.
3. In the main function, the number of refinement steps is already set to converge to a quadratic B-spline. Compile your code, now the two curves (white and red) must overlap.

Doo-Sabin subdivision scheme

Doo-Sabin's scheme is a surface subdivision algorithm. The smooth surface is computed iteratively from initial control points by calculating several "patches" which are then assembled (Theory: ppt 5, p. 57-62).

1. Compile. In the FLTK window: the red surface is the B-spline surface the Doo-Sabin's subdivision algorithm must converge to.
2. In the file "`api/doosabin.cc`", implement the three following functions:

   - `void refine_once()`
     Its purpose is to perform one refinement step with the Doo-Sabin's algorithm. It contains already some code: a double loop which runs through all the 3x3 patches of control points of the current surface. The steps in this loop are:

     i. `extract_3x3_patch` returns the grid of 3x3 control points of the current "patch" into the variable `patch`. This function is already implemented.

     ii. `sub_3x3_patch` (**to be implemented**) computes a 3x3 "sub-patch" given a 3x3 "patch" of control points and two matrices `S1` and `S2`.

     iii. `sew_sub_patches` (**to be implemented**) "sews" together the "sub-patches" `Q00`, `Q01`, `Q10` and `Q11` into a single 4x4 refined "patch" and stores it into the variables `refined_patch`, which is a 4x4 grid of points.

   - `void sub_3x3_patch(const npoint patch[3][3], const Square_matrix &S1, const Square_Matrix &S2, npoint Q[3][3])`

`const`

Given a "patch" of 3x3 = 9 control points, it computes the control points `Q` of one of the four "sub-patches" corresponding to each quadrant of the initial "patch", where:

i. `patch` is a 3x3 matrix of control points.

ii. `S1`, `S2` denotes respectively the left and right 3x3 matrices multiplying the matrix of control points (see, e.g., lecture 5, p. 61). Use the `Mult` function of class `Square_Matrix` in file "`nutil/linear_algebra`" for performing matrix multiplications.

iii. Following the values of `S1` and `S2`, `sub_3x3_patch` will compute the control points `Q` of on "sub-patch" corresponding to a given quadrant of `patch`.

iv. The new control points of the current "sub-patch" have to be returned in variable `Q`, which is a 3x3 grid of `npoint`.

- `void sew_sub_patches(const npoint Q00[3][3], const npoint Q01[3][3], const npoint Q10[3][3], const npoint Q11[3][3], npoint rpatch[4][4]) const`

  It assembles four 3x3 "sub-patches" `Q00`, `Q01`, `Q10` and `Q11` into one refined 4x4 "patch" of points.

  i. `Q00`, `Q01`, `Q10` and `Q11` 3x3 "sub-patches" corresponding to the quadrants of an initial 3x3 "patch" (see figure 1). These "sub-patches" are computed by the above `sub_3x3_patch` function.

  ii. `rpatch` is the 4x4 grid of points resulting from the assembly of the 3x3 "sub-patches" `Q00`, `Q01`, `Q10` and `Q11`.

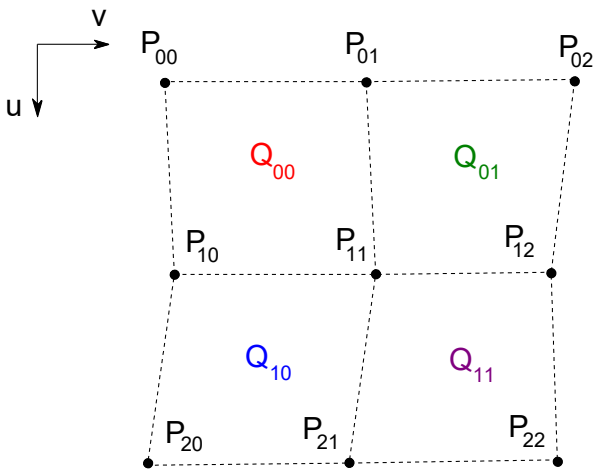

Figure 1.

As usual, files to be sent the evening before next session (or before) at a.bolyn@uliege.be :

- chaikin.cc
- doosabin.cc