Assignment in Computer Graphics II
– Assignment 7 –

Computer Graphics and Multimedia Systems Group
David Bulczak, Christoph Schikora

Assignment 1 [1+1 Points] Programming: Subdivision Curves

Download the program framework `assignment_07.zip` from the practice website. The aim of the task is to implement the two subdivision-algorithms Chaikin and 4-Point Subdivision.

One part of the task is optional. You can earn one extra point for programing the other part to supplement your point account. For the 4-Point scheme part also the additional question has to be answered.

The available program has the following functionality:

1. Read data as an argument (eg data_01): Number of polygon vertices n and the corresponding vertices.
2. With the `'+'` and `'-'` keys the level of subdivision can be adjusted.
3. With the keys `'('` and `')'`, the weight w of the 4-Point-Subdivision scheme can be configured.

The provided program consisting of the following classes:

`P3D, V3D`: A 3D point and vector class incl. of various operators to manipulate points and vectors.

`Polygon3D`: The already defined class of 3D polygons.

`Subdivision`: Class for the calculation of subdivisions for a given polygon `Polygon3D`.

`DisplayOGL`: Class for displaying the input polygon and subdivision curves.

The class hierarchy is complete, however, lacks the implementation of methods for subdivision.

1. Implement the following methods **recursively**
   - `static Polygon3D computeChaikinSubdivision(unsigned int level);`
   - `static Polygon3D compute4PointSubdivision(unsigned int level, double w);`
2. What happens if weight w is chosen to be greater than 0.5? How do you explain this behavior?

Assignment 2 [1 Point] Solid Modeling

To achieve boolean operations with polygonal b-reps, efficient computations for polygon intersections are necessary.

Given two planar polygons with vertices $\{\mathbf{P}_1,\dots,\mathbf{P}_k\}$ respectively $\{\mathbf{Q}_1,\dots,\mathbf{Q}_l\}$ and two planes E_P, E_Q containing the polygons.

E_P, E_Q are given in Hesse normal form:

$$E_P: \quad \hat{\mathbf{n}}_P \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + d_P = 0, \quad \hat{\mathbf{n}}_P = \begin{pmatrix} a_P \\ b_P \\ c_P \end{pmatrix}$$

$$E_Q: \quad \hat{\mathbf{n}}_Q \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + d_Q = 0, \quad \hat{\mathbf{n}}_Q = \begin{pmatrix} a_Q \\ b_Q \\ c_Q \end{pmatrix}$$

1. Show that for arbitrary vectors

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \ \mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

   the following holds:

$$\mathbf{b} \times \mathbf{a} = -\mathbf{a} \times \mathbf{b}, \tag{1}$$

$$\mathbf{a} \times \mathbf{a} = 0, \tag{2}$$

$$\mathbf{a} \cdot (\mathbf{a} \times \mathbf{b}) = 0, \tag{3}$$

$$\mathbf{a} \times (\beta\mathbf{b} + \gamma\mathbf{c}) = \beta(\mathbf{a} \times \mathbf{b}) + \gamma(\mathbf{a} \times \mathbf{c}) \tag{4}$$

   with scalar factors $\beta,\gamma \in \mathbb{R}$.

2. To implement the intersection computation we need a function `intersectPlane`, that intersects two planes E_P, E_Q and returns a line G in parametric form.
   Proof that

$$G: \mathbf{P} + \alpha\vec{\mathbf{l}}, \alpha \in \mathbb{R} \text{ with } \mathbf{P} = \frac{(d_Q\hat{\mathbf{n}}_P - d_P\hat{\mathbf{n}}_Q) \times (\hat{\mathbf{n}}_P \times \hat{\mathbf{n}}_Q)}{\|\hat{\mathbf{n}}_P \times \hat{\mathbf{n}}_Q\|^2} \text{ and } \vec{\mathbf{l}} = \hat{\mathbf{n}}_P \times \hat{\mathbf{n}}_Q$$

   solves the problem!

   Hint:

   1. **Think about the properties of G w.r.t. the planes.**
   2. $\hat{\mathbf{a}} \cdot (\hat{\mathbf{b}} \times (\hat{\mathbf{c}} \times \hat{\mathbf{d}})) = (\hat{\mathbf{a}} \times \hat{\mathbf{b}}) \cdot (\hat{\mathbf{c}} \times \hat{\mathbf{d}})$

Assignment 3 [1 Point] L-Systems

Given the alphabet $V = \{\Delta, s, +, -, [,]\}$, the axiom $\omega_0 = \Delta$ and the rule

$$p(\Delta) = s[\Delta][+\Delta][-\Delta], \quad \text{else } p(x) = x, \forall x \neq \Delta$$

The geometrical interpretation of a word is implemented in an OpenGL program in the following way:

character		**OpenGL-Code**
Δ	$\longrightarrow$	`glBegin(GL_TRIANGLES);`
		`glVertex2f(0.0, 0.0);`
		`glVertex2f(1.0, 0.0);`
		`glVertex2f(0.5, sqrt(3.0)/2.0);`
		`glEnd();`
s	$\longrightarrow$	`glScalef(0.5, 0.5, 1.0);`
+	$\longrightarrow$	`glTranslatef(1.0, 0.0, 0.0);`
-	$\longrightarrow$	`glTranslatef(0.5, sqrt(3.0)/2.0, 0.0);`
[	$\longrightarrow$	`glPushMatrix();`
]	$\longrightarrow$	`glPopMatrix();`

Sketch the figures you get after 0-, 1-, 2- and 3- iterations.

Submission: 27.11.2014, before /at the beginning of the exercise.

Submit task 2 and 3 (also task 1 additional question) on paper and send for task 1 an email with the modified files.
$\rightarrow$ Email to: david.bulczak@uni-siegen.de, christoph.schikora@uni-siegen.de

The **deadline** is the same for all tasks, e.g. emails will only be accepted till Thursday 12:00 clock. If we receive your mail, we will send you as soon as possible a confirmation.