Assignment no. 3

due: June 1st, 2005

The topic of this assignment is Minkowski sums of convex polyhedra. For two sets P and Q in $\mathbb{R}^3$, their Minkowski sum denoted $P \oplus Q$ is the set $\{p + q | p \in P, q \in Q\}$.
Each of the following exercises should result in a procedure that needs to be plugged into the marked point in the file `viewer.cpp` which available from the auxiliary assignment web-page. See more about the viewer below.

Exercise 3.1: Minkowski sums of a collection of polytopes Implement a global function-template that computes the Minkowski sum of a range of convex Polyhedra in $\mathbb{R}^3$.

```
  template <class T_polyhedron_iterator, class T_polyhedron>
  void minkowski_sum(T_polyhedron_iterator begin, T_polyhedron_iterator end,
                     class T_polyhedron & polyhedron);
```

The function accepts three parameters. The first two define an input range of convex polyhedra. The third is a reference to the resulting Minkowski sum. An implicit precondition is that the template parameter `T_polyhedron` must be the value type of the template iterator `T_polyhedron_iterator`. Use the CGAL `Polyhedron_3` data structure to represent a convex polyhedron.
Let $P_1, \ldots, P_k$ be the input set of polyhedra, and let V_i be the set of vertices of P_i. A simple way to compute the Minkowski sum of a set of convex polyhedra is based on its equivalence with the convex hull of the following set of points: $\cup_{v_1 \in V_1, \ldots, v_k \in V_k} \{v_1 + \ldots + v_k\}$.

Exercise 3.2: Morphing between two convex polyhedra We wish to morph between two given convex polyhedra P_0 and P_1. The intermediate shape $Q(t)$ for $t \in [0,1]$ is defined as $Q(t) = (1-t)P_0 \oplus tP_1$, where tP_1 for example is the polyhedron obtained by scaling each vertex of P_0 by t. Clearly $Q(0)$ equals P_0 and $Q(1)$ equals P_1.
(a) The key to obtaining an efficient and relatively simple procedure for this type of morphing is understanding the combinatorial structure of the family of polytopes $Q(t)$ for $t \in (0,1)$. Characterize the polytopes $Q(t)$ in this family. In order to do this notice that the normal diagram (or Gaussian map) of the Minkowski sum is the result of overlaying the normal diagrams of the two summands. For more information and explanations, see:
http://www.cs.tau.ac.il/~efif/publications/exact_mink_3d/exact_mink_3d.pdf
(b) Implement the function `Morph_polyhedra` that accepts two convex polyhedra P_0 and P_1 and a resolution parameter $\delta = \frac{1}{n}$. Each call to the function it produces the next morph $Q(i\delta)$ for $i = 0, \ldots, n$.

```
template <class T_polyhedron>
class Morph_polyhedra {
private:
  T_polyhedron & m_summand1;
  T_polyhedron & m_summand2;
  T_polyhedron m_minkowski_sum;
  bool m_dirty;

public:
  Morph_polyhedra(T_polyhedron & p1, T_polyhedron & p2) :
    m_summand1(p1), m_summand2(p2), m_dirty(true) {}

  T_polyhedron & operator()(float t) {....}
}
```

A Ready-Made Viewer

An application that allows you to view multiple polyhedra was created for your convenience. Use this ready-made viewer to produce an animation that shows all the morphs in succession. The application details follow.

The application parses multiple ASCII files provided in the command line. Each file describes a convex polyhedron in a format very similar to VRML, but you really don't have to bother with details. It opens a window, creates a graphics context, computes the viewing parameters so that all the polyhedra are visible, and renders them into the window.

The application uses a class called `Polyhedron_viewer`. It contains a data member of type CGAL `Polyhedron_3`, and a few other functions. For example:

- `parse(char * filename)` — parses the given file.
- `draw()` — draws the polyhedron.
- `update()` — updates the `Polyhedron_3` internal structure.

For each input file the application creates an instance of `Polyhedron_viewer` and pushes it into a global container of `Polyhedron_viewer`'s called `s_polyhedrons`. Then, the application computes the bounding sphere of all polyhedra stored in this container, and uses it to compute the point-of-view and viewing frustum of the graphic context. Finally, it renders all the polyhedra stored in the container `s_polyhedrons` onto the window, and is suspended, unless the '-m' option is provided on the command line. In this case, the application is not suspended. Instead, the function 'idle' is invoked in a cycle. Use it to apply the morphing on the polyhedra.

You don't need to bother with the parsing of the input files, as the `parse()` function does the job, but in case you would like to create your own input, the description of the syntax is given in the auxiliary assignment site.