Applied Computational Geometry — Spring 2009 — Dan Halperin

Assignment no. 4

due: Thursday, June 11th, 2009

This assignment has been prepared by Efi Fogel, `efif@post.tau.ac.il`. Feel free to contact Efi for advice and assistance.

The topic of this assignment is Minkowski sums of polytopes. For two polytopes P and Q in $\mathbb{R}^3$, their Minkowski sum denoted $P \oplus Q$ is the set $\{p + q \,|\, p \in P, q \in Q\}$.

Exercise 4.1: Width of polytopes (50 points)
Let P be a set of point in $\mathbb{R}^3$. The *width* of P is the minimal distance between a pair of parallel planes such that the closed region between the planes contains P.
Develop code that computes the squared width of a polytope using three different methods described below. Compare the performance of the different methods on various input polytopes. Could you explain the source of the differences?
The three methods are:

1. Computing the width directly using `CGAL::Width_3`.
2. Computing the width using Minkowski sums, where Minkowski sums are computed using convex hulls (`CGAL::convex_hull_3`).
3. Computing the width using Minkowski sums, where Minkowski sums are computed using (spherical) Gaussian maps (`CGAL::Arr_polyhedral_sgm`).

Implement an application called `width` that accepts a name of a file as a single argument in the command line. The input file describes a polytope in a format very similar to Vrml, but you really don't have to bother with details. The application parses the input file, applies the methods above sequentially, verifies that the computed values are identical, and prints out the results.
Assuming a file called `tetrahedron.wrl` resides in the current directory, typing

```
width tetrahedron.wrl
```

should result with:

```
tetrahedron sd t1 t2 t3
```

where `sd` is the squared width of the polytope described by the input file `tetrahedron.wrl`, and `t1`, `t2`, and `t3` are the number of seconds it took to compute the squared width using the three methods above, respectively. All numbers are real.

Facilitating the parsing of input files

Use the lexical and syntactical parsers provided as part of the `viewer` application presented below to parse files that describe polytopes. Additional information regarding compilation instructions and input files is provided in the course web-page.

Exercise 4.2: Morphing between two polytopes (50 points)
Develop an application that morphs between two given polytopes P_0 and P_1 in $\mathbb{R}^3$. The intermediate shape $M(t)$ for $t \in [0,1]$ is defined as $M(t) = (1-t)P_0 \oplus tP_1$, where $tP = \{tp \,|\, p \in P\}$ is the result of scaling P. Clearly, $M(0) = P_0$ and $M(1) = P_1$.
The key idea behind an efficient procedure for this type of morphing is exploiting the structure of the family of polytopes $\{M(t) \,|\, t \in (0,1)\}$. Recall that (i) the Gaussian map (also referred to as the normal diagram) of the Minkowski sum of two polytopes is the overlay of the Gaussian maps of the two summands, and (ii) the combinatorial structure of the Gaussian maps of the two summands does not change under scaling.
Implement a function object called `Morph_polytope`. Its constructor accepts references to three polytopes P_0, P_1, and M, where M is a placeholder for the Minkowski sum $P_0 \oplus P_1$. Each call to the member function `operator()(t)` of `Morph_polytope` should result with $M(t)$. Plug the function object `Morph_polytope` in the viewer application described below.

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 files provided in the command line. Each file describes a polytope in a format very similar to VRML. It opens a window, creates a graphics context, computes the viewing parameters so that all the polytopes 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` that represents a polytope, and a few other functions, for example:

- `parse(char * filename)` — parses the given file.
- `draw()` — draws the polytope.
- `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_polyhedra`. 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 polytopes stored in the container `s_polyhedra` onto the window, and is suspended, unless the '-m 1' 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. Additional information regarding compilation instructions and input files is provided in the course web-page.

Exercise 4.3: Gaussian maps, optional (20 points)
(a) Show that not every convex subdivision on a sphere induced by geodesic arcs of length strictly less than 180° degrees is a valid Gaussian map.
(b) Consider the cubical Gaussian map data structure.

- How many planar maps (of the 6) at most contain faces which are the mapping of a single vertex of a polytope? Show an example.
- How many planar maps (of the 6) at most contain edges which are the mapping of a single edge of a polytope? Show an example.